Web Security and Privacy notes

Salvino Matteo

Contents

1 Email security

The mail system is made up of several components and aspects, it has a basic architecture and functioning, then the Internet community started to add some extensions (MIME) and enhance the basic functionalities adding security standards such as SPF or DKIM. Then, we will dedicate our time to understand if and how an email has been forged in order to trick the receiver in doing something wrong. Finally, we will spend some time talking about how to secure email, we will talk about PGP, which is possibly one of the first proposal to secure the mail system.

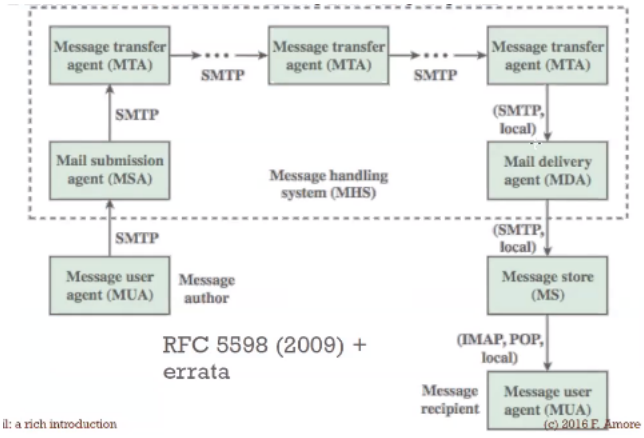
1.1 E-Mail architecture

The email service is something we are using today by access a very simplified client that hides a lot of complexity of the underlying system. In particular, the e-mail architecture is constituted by several components :

- **Message user agent (MUA)** : it's actually the client used by end user.
- **Mail submission agent (MSA)** : it sends the message to the recipient using a MTA. It interacts with the MUA to receive commands, store locally the information that are then passed to the MTA to create the real email.
- **Message transfer agent (MTA)** : it's what we typically call the mail or SMTP server. In general, we can have several MTA in order to reach the MTA of the email recipient. When the email arrives to such MTA, it recognize that it doesn't need to relay the message anymore and deliver the message to a MDA.
- **Mail delivery agent (MDA)** : it can possibly store the message on a MS. Typically, the MDA and MS are implemented by a single process.
- **Message store (MS)** : it's a location where the MDA can store the message received from the sender.
- **Message user agent (MUA)** : once the message is stored, it can be read or retrieved by the user using its client.

The interaction between MS and MUA is quite complex and it can be done through different approaches. The easiest case is the local approach, where

the user logs in on the machine that is hosting the message store and reads directly the message on the MS. The more common choice is to use a protocol to read email using a client that will fetch the email from the server and make possible for the user to read that email on a remote machine. This typically happens through different protocols such as **POP** and **IMAP**. With the first protocol, our client contact the MS and say "give me a copy of all emails received from this date to this date". Next, it retrieves such emails and then they are deleted from the MS (there is also an option to leave the email there). The concept of IMAP is quite different; in fact the MS is designed to store emails. When we access the MS to read emails, we ask it to make a copy of a specific message we want to read. If we want to move emails or performs any other operation, we will sends commands to the MS, but the operation is performed on the MS. What we see with IMAP on our client, is somewhat a view of our message box on the MS. This is why the client need to use the protocol to synchronize its local view with respect to the reality that is given by the status of the MS. MTAs speaks between themselves using the SMTP protocol. Next, this protocol was extended to include some enhanced features such as encryption, which goes under the name of **ESMTP**. This extension is designed such that all software compatible with ESMTP will be also compatible with SMTP. SMTP exchanges messages using the message envelope separate from the message itself. The protocol also defines the way sender and receiver can be identified, typically using the standard email addresses. To have in mind the previous architecture we report the following picture.



In general, the way emails are transferred on the Internet follows the concept of the **store-and-forward model**. This means that, every time a MTA needs to send a message to the next one, it first stores the message locally

and then forward the message to the next one waiting for a confirmation of the recipient (only at that point can delete the local message if needed). This enables one of the main characteristics of the system, which is the fact of being asynchronous. Typically an email is made up of three components :

- **message envelope** : it's used by MTAs to correctly relate the message from the sender MTA to the recipient MTA.
- **message header** : where we write who is sending the email, who is the recipient, i.e. they are meta information. In particular, it's structured into fields such as From, To, CC, Subject, Date and other information about the email. It's separated from the message body by a blank line.
- **message body** : it's the text of the message that we want to send, typically unstructured.

Each message has exactly one header, which is structured into fields. These fields are structured with name and value that are separated by a double column. There is also a possibility to define multi-line header fields. There are mandatory header fields such as :

- **From** : it's the sender email address.
- **Date** : it's the local time and date when the message was sent.
- **Message-ID** : it's an automatically generated field by the sender, to uniquely identify the message.
- **In-Reply-To** : it's a message-ID that represents a link to a previous message to which the current message is a response to.
- **To** : it's the recipient email address.

Other common header fields are the following :

- **Subject** : it's a brief summary of the topic of the message.
- **Bcc** : with this option we can add addresses to the SMTP delivery list but they will not appear in the message data, i.e. they are invisible to other recipients.
- **Cc** : it's similar to the previous field, but in this case all the addresses in the SMTP delivery list are listed in the message data.

- **Content-Type** : it's the field that allows the use of extensions like MIME type.
- **References** : it's similar to the In-Reply-To field, but contains a list of previous messages. It should be coherent with In-Reply-To field.
- **Reply-To** : it's mainly used to indicate to the client what to do if the user will click the reply button.
- **Sender** : it's the address of the actual sender acting on behalf of the author listed in the From field (list manager, secretary, etc).

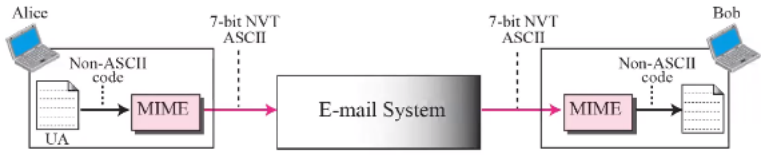
Together with the headers that we talked about, SMTP defines a lot of trace information that are included in the message as it's relayed toward the destination, which are also saved in the header using the following two fields :

- **Received** : it's populated every time the message is received by a MTA, with information about the MTA itself such as location, IP address, domain, etc. In this way we can rebuild the history of the message through the various relaying steps in its path toward the destination.
- **Return-Path** : it's a field defined by the final MTA.

1.2 MIME

Another standard often used in email system is MIME (Multipurpose Internet Mail Extensions), but it's also used in the web for other reasons. It's a standard that explain us how to encode complex content in more simpler containers. It allows to include : text in character sets other than ASCII, non-text attachments, message bodies with multiple parts and header information in non-ASCII character sets. MIME is one of the things that changed the usage of Internet in the last 20 years. Now, the question is how does MIME works ? The idea is the following one : when the user want to send a message through the email system that contains some information that are not in the ASCII character set, the client encodes this information using MIME in a transparent way. First it analyze the content that is provided by the user, checks how each part of the content should be encoded and set up a MIME compliant message with the correct encoding of each part. Then this MIME encoded message is fully expressed using only 7 bit ASCII encoding and then it can be sent via the email system. At the other end of the email

system when the message is received, the client of the receiver apply the opposite process, decoding the MIME content and correctly rendering the content of the message.



NVT = network virtual terminal

The important point is that the MIME encoding of the message body is completely transparent to the email system, because MIME provides a way to encode and decode this content at the endpoint, and everything is completely transparent to the email system since it never look at the message content. The kind of encoding used for MIME is defined through MIME headers which contains the version information, the type of the content that will be provided by the email body and other meta data such as encoding type, content-id and content-description. Some of the most common MIME types are *text/plain* for plain text data, *text/html* for html content, etc. If for example we consider a jpeg image, this is not basic ASCII, how can we represent this richer content within a part of the body with the correct content type, but still sticking with the standard 7 bit ASCII ? For this reason, MIME defines several content transfer encoding methods : 7 bit, 8 bit, binary, base64 and quoted-printable encoding. Base64 is the standard way to encode non textual content in MIME messages. The idea is that through base64 we start from a binary content and obtain as target a string that is limited to $A-Z, a-z, 0-9, +, /$ character set. Typically the Non-ASCII data are divided in group of 8 bits, and combined in a certain number of 6 bit chunks. After that, each of them is interpreted as a number that is used as an index towards the 7 bit ASCII table. The good point is that having chunks of 6 bits we can range from 0 up to 63, which correctly maps to a 7 bit ASCII table. Considering that we are splitting and combining back the original stream of data, we may end up with some missing characters. So base64 encoded data typically may end with one or two symbols (they are equal symbols). The first advantage of this encoding scheme is for sure its simplicity to implement and apply to any kind of byte encoded content. Another good point is that, once we have the possible indexes that comes up from the various combinations of 6 bit that we use as chunk size, we can easily build a base64 converting table, such that an algorithm can just work

as an iteration on the byte stream that looks up characters that make up the encoded stream. The disadvantage is that we are just using a subset of the 7 bit ASCII character set, and this means that there is some space that is wasted in the encoding. The encoding is obviously not space efficient. Another option is called quoted-printable encoding, where any 8 bit byte value may be encoded with three characters : an equal sign followed by two hexadecimal digits representing the byte's numerical value; instead any non 8 bit byte values are ASCII chars from 33 to 126, excluding 61 (equal sign). It's extremely inefficient, because we encode 8 bits with 21 bits. However, its main advantage is that if for some reason the endpoint that receive the message is not able to decode quoted-printable encoding, the text is still mostly readable.

1.3 Unwanted email messages

Lets try to understand which are the weaknesses of these standards and how they are used to send unwanted email that typically comes with the name of **SPAM**. They are used to advertise any kind of merchandise. It's also used for a different goal such as performing some actions to take control of part or the full machine that the user is actually connected with. Another source of spam are called **e-mail chain letters**, with which induces you to fraud other people to get some benefit. Another characteristic of spam is the quality of the message itself. For example a message with a very low text quality, can be easily classified as spam. However, the low quality is something that is disappearing quite quickly. Naturally the spammers main goal is to spread the malware, in order to control computers for future attacks, steal sensitive information, and so on so forth.

1.4 Basic email nonalogue

We report a brief list of suggestions to follow when using the email system.

- Disable html message or, at least, disable download of remote images
- Don't click links, since we could redirect us to bad web sites containing malware
- Don't open unknown attachments since they may contain malware
- activate local anti-spam filter
- Don't participate with chain letters

- Protect and respect privacy of other recipients
- Even if non-Windows user, activate antivirus for protecting your recipients
- Don't provide your personal data
- Don't click "delete me" link.

1.5 SMTP extensions

We know that there are some pitfalls in the email system. For this reasons, there are other solutions that complements SMTP in order to provide some security features. In particular we will talk about three of them : **SPF**, **DKIM** and **VBR**.

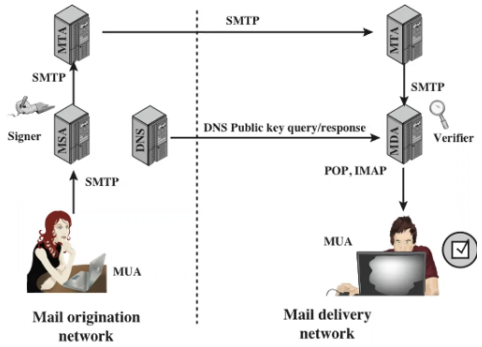
1.5.1 Sender Policy Framework

One big problems with emails is the fact that the body and the envelope of the email are somewhat separated entities. The body of the email is built in the client that creates the email, while the email envelope is forged on the server through the interaction between the client and the server. Furthermore, in the SMTP protocol no where is written that it should be a coherence between some of the headers present in the email and what is written in the envelope (e.g. From field in the body equal to the From field in the envelope). SPF is a standard to solve this problem from a specific standpoint. In particular, SPF protects the sender address only in the envelope, it doesn't care about the From field in the email header. It allows the administration of the domain through which an email is sent, to define some policies about who could send emails through that server. So, receivers verifying the SPF records may reject message from unauthorized sources before receiving the body of the message. The nice point is that this will happen only looking at the email envelope; so is the server itself that can do this check and possibly reject the message, without the client even receiving the body. When the server accept the sender, it receives through the SMTP interaction the list of recipients and the message body. At that point it's expected to insert in the message header a field called Return-Path, in which it needs to save the sender address as is reported in the envelope. This means that this Return-Path may or may not correspond to the From header in the message. If the two are the same typically the message is fine; if they are different, SPF does not necessary mark the message as spam.

We want to underline that SPF doesn't enforce the correspondence between these headers; they should match, but not in all cases. Naturally SPF by doing checks only on the sender field in the envelope has some limitations : keep the SPF record update as companies change service providers and add mail streams is difficult. Another bad point is that given that SPF is not perfect, the check on SPF it's just one among several checks that email servers typically do to avoid spam to reach a target email. Another important point is that typically SPF breaks whenever a message is forwarded, because the forwarder address could not be whitelisted in the DNS text record and at that point the SPF check will fail. The last point is that SPF makes no check on the From header contained in a message that wrt SPF can be easily spoofed.

1.5.2 DKIM

Given that SPF only works on the email envelope, someone started to think about introducing a different solution to take care of the security of the message headers called **DomainKeys Identified Mail**. It specifies how some parts of the message can be cryptographically signed in order to avoid their content to be tampered by other people (MITM attack or non-intended source creates a fake message with false source information). The idea of DKIM is that when you receive a DKIM signed message, as a recipient you can verify the signature that is contained in the message by querying the domain dns server for the information needed to check that signature. This is important because the rational behind DKIM is that it links the content of the email that is signed, with an information that is obtainable through a query to the dns of the sender. So the idea is that the two will be coherent only if they originates from the same person (domain administrator) or people that the domain administrator allows to use this kind of feature. A classic deployment of DKIM works in the following way



First of all we have the origin of the email through a client. The client connects through SMTP with the server, but the call is intercepted by a **message signing authority (MSA)** that provides all the features to correctly sign the content of the message. Then the MTA through the SMTP protocol delivers the message to the endpoint to the server on the recipient side and this goes through a MDA, which checks through a query to the dns system if the signed content of the message is verified or not. If it's verified the MDA delivers the message to the end user. However, there is a practical problem when this protocol is widely deployed on the web. The email may pass through several servers, that typically may decide to add information in the headers of the email to keep track of what happened to the message on its path to the destination. Some of this changes can easily break the signature, because even if we just include the headers that are not expected to change in the path, some servers may include processing features that will alter the content of those headers. To avoid this kind of problem DKIM include a mechanism called **canonicalization**, through which it's possible to tell to the recipient if he needs to be strict in the checking of the signature of if he can perform a relaxed check. In this latter case, before applying the hash function to the headers, the recipient will for example make all the headers lowercase, remove all extra spaces, etc. Typically the relaxed checks says from easy rejection due to somewhat changes in the header that don't represent real mangling with the message content. Another detail that is important is represented by **selectors**. They allow to multiplex several private/public key pairs for the same domain, while providing a way for the recipient to check the correct one.

1.5.3 DMARC

In order to make SPF and DKIM work together to guarantee us that no one will make a wrong use of the available email services, we can use a standard called **Domain-based Message Authentication, Reporting, and Conformance (DMARC)**. It's interesting because it's not really a technical standard, but it's more or less a standard defining a process to integrate SPF and DKIM within a set of best practices that allow to make the best of them. In particular, DMARC allows organization to publicly declare policies through which they can define which kind of practices they put in place to guarantee email authentication. Furthermore, it provides instructions to the recipients to enforce these policies and report possible misuse. In particular, the domain owner publish its practices, he tell what to do when there is a failure with an authentication check and enables reporting.

How does DMARC works ?

1. The first thing you need to do in order to use DMARC is to publish a policy. The publishing of a policy is performed through your dns records and again it uses the TXT records embedding some information for DMARC associated with that specific domain. In particular, it uses a TXT record that starts with *_dmarc.* followed by the name of your domain.

2. When the recipient server receives an incoming email containing the DMARC headers, it performs first of all a dns query to check the policies of the sender. This query is based on the domain declared in the From field of the email. Next, it checks three aspects of the message : checks if the message's DKIM is valid or not, checks if the message come from IP addresses allowed by domain's SPF record and checks if the message headers show proper domain alignment. The domain alignment is an aspect that is specific of DMARC. It expands what SPF and DKIM actually already do, and it checks if various elements in the domain headers match the domain declared in the From field. In particular, for SPF it checks if the domain in the From field correspond to the domain indicated in the Return-Path field; for DKIM it checks if the domain in the From field correspond to the domain listed in the d field of the DKIM header.

3. With this information, the recipient server is ready to apply the sending domain's DMARC policy to decide whether accept, reject or flag the email message.

4. Finally, the recipient server will report the outcome to the sending domain owner. In particular, there are two formats of DMARC reports :

    - **Aggregate reports** : they represent statistical data on how many spoofed email for those domain has been received on a total of emails, or stuff like that. They are sent periodically, and not immediately sent at the reception of an email.
    - **Forensic reports** : they represent information about the checks the recipient server has performed and why they failed, but it has to include in the reports the original email that failed the checks. It allows the sender to perform some debugging for example to see if there is a problem with the DMARC configuration and it also

allow the owner of the sender domain to check if spoofing happens with specific patterns and possibly track down the origin of those spoofed emails and take some mitigations for that.

In particular a DMARC record has the following fields :

- **v** : it specifies the DMARC version to be used.
- **p** : it specifies which are the standard policies that needs to be applied to emails. It can assume one of the following three different choices : **none** for treat the mail the same as it would be without any DMARC validation, **quarantine** for accepting the mail, but place it somewhere other than the recipient's inbox, and **reject** in order to reject the message.
- **rua** : it's the mail address to which aggregate reports should be sent
- **ruf** : it's the mail address to which forensics reports should be sent
- **pct** : it specifies the percentage of mail to which the domain owner would like to have its policy applied (it's an optional parameter).

The outcomes from protocol checks are reported in the mail headers.

1.5.4 VBR

Another protocol that we want to talk about is called **Vouch By Reference (VBR)**. It takes a completely different approach that is still orthogonal to SPF and DKIM, and in fact it can be used together with these solutions. In particular, VBR allows the implementation of sender certification by third-party entities. The idea is that it allows providers to independently certify the reputation of senders by checking the domain name. When you send an email using VBR, you use the standard DKIM protocol to sign the email, and then include in the signature a VBR-info field, which includes several information such as the domain name that is going to be certified, which content of the message is allowed to deliver and a list of one or more vouching services, that is the domain names of the services that vouch for the sender for that kind of content. At the receiver side, after checking the correctness of the DKIM or SPF headers, the software may possibly double check the VBR info by performing dns queries on the domain name of the services included in the VBR-info field. In particular, it will look for entries of type TXT that have this structure *domain name._vouching service*. If VBR is checked at recipient side, the outcome of this check will be included in the Authentication Results field in the message header.

1.6 Basic email analysis

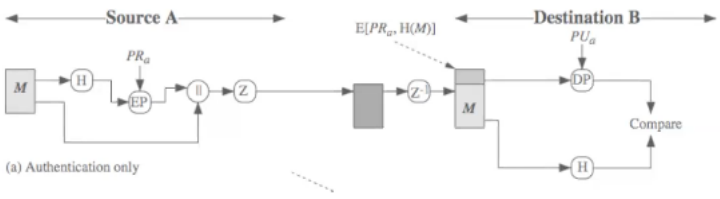
Email spoofing means altering some or majority of the elements that are present in the email header. The spoofer do that for hiding its real identity, since most of the content of spoofed emails is typically malicious in some form. How can the content of the email headers be spoofed ? At least for the sender address this is really simple, since SMTP doesn't include any authentication for senders. How do we check for spoofing ? Actually there are no deterministic approaches for doing that. For this reason, we can follow some guidelines that helps to understand if the message has been spoofed or not. A good starting point to do that is to analyze the complete message (full header + body), checking the From, Return-Path, Reply-To and Received header fields. In particular, in order to implement a secure email service we need :

- **confidentiality** : protection of the content from disclosure
- **authentication** of the sender of the message
- **message integrity** : protection from alterations
- **non-repudiation** of origin : protection from denial by sender.

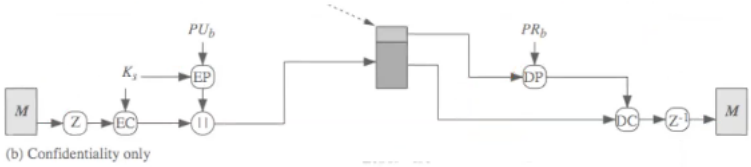
1.7 PGP

Pretty Good Privacy (PGP) is an open protocol with proprietary and publicly open implementation. The basic point about PGP is that it was designed by incapsulating the usage of cryptographic primitives inside the email usage process. PGP is constituted by two main functionalities :

PGP Authentication When the sender creates the message, computes its hash and uses a private key to sign the hash. Next, the message and the signature are put together to form the signed message. On the recipient side, the message signature is decoded using the sender public key and the result is compared with the hash of the message itself. If the two hashes are coherent this means that the message has not been tampered.



PGP Confidentiality First the message is compressed and encrypted by the sender using a symmetric key called **session key**. Next, the session key is encrypted using RSA with the recipient public key. Next, the encrypted session key and encrypted message are sent together. The recipient takes the encrypted session key and decodes it using its own private key, obtaining the session key, that will be used to decode the original message.



(b) Confidentiality only

Of course these functionalities can be used together (confidentiality & authentication). As previously mentioned, PGP uses a compression function to reduce the size of messages, typically before the encryption phase. In this way it's possible to store uncompressed message and signature for later verification, because the point is that compression algorithm may introduce some non determinism. When we use PGP for sending binary data, it will encodes such data into printable ASCII characters using Base64 encoding. Session keys that are used for encode messages are built randomly for each encryption pass, and the implementation is typically based on a 256 bit buffer which contains random bits. This buffer is updated every time you press a key interacting with PGP. It register the period of time between two keystrokes, plus the value of the keystroke itself, and use this information as a key to encrypt again the content of that buffer. In PGP you may possible own as a user more than one public/private keys pair, in order to send different kind of messages or to interact with a different group of people. To identify which couple has been used to encrypt a message, the message is sent including the public key plus the ID of the key identifier of that couple of key. The key identifier is simply extracted from the least significant 64 bits of the public key. In PGP implementation each user maintains through its client two **key rings** : the public key ring contains all the public keys of other PGP users known to this user, indexed by the key ID, the private key rings contains the public/private key pairs used by the current user and they are indexed by the key ID and encrypted using some passphrase. Now the question is, how do we manage these keys ? In PGP every user represent its own Certification Authority (CA), which is able to create all the public/private key pairs needed. The main disadvantage is that this removes the so called assumption on the presence of a trusted third party, that makes this approach usable for example for signing digital certificates.

Instead, PGP rely on the concept of **web of trust**. The idea is that when you start using PGP, you need to get the recipients public keys, and then start to trust these recipients and they will trust you. This mechanism will iteratively creates a web of trust, where keys brings with them a sort of certificate of trust that says that those keys have been trusted by other users and this should increase the possibility that they are also trusted by someone else that have never seen before those keys. For this reason, key rings includes trust indicators in their data structure. This web of trust somewhat is a vision that looks at reality where the trust is something that emerges from a decentralized fault-tolerance web of confidence.

1.8 ARC

The **Authenticated Received Chain (ARC)** protocol proposes a solution to overcome the problems that we may have using SPF and DKIM. In particular, instead of enforcing security only between the sender and receiver, it allows to completely track security checks through the full chain of relays that are encountered while an email is in transit. The reasons why messages can fail DMARC policies checks are for examples : strict policies, the message pass through some indirect mail flow such as mailing list, attachments removal, etc. To overcome these problems people started to think about ARC trying to sketch out which are the main design choices for the definition of this new protocol. In particular, the main design decisions are the following :

- the originator of the message doesn't need to make any change to the message itself
- convey Authentication-Results content intact from the first ARC intermediary forward
- allow for multiple hops in the indirect mail flow
- ARC headers can be verified at each hop
- work at Internet scale
- we need to make ARC as independent as possible from DMARC.

The idea of ARC on the recipient side is that the receiving endpoint should perform the standard DMARC checks. But if a failure is result of these checks, then optionally the receiver may possibly look for the presence of ARC headers, and use them to perform a more extensive check. If this

extensive check give us a negative result then the server may possibly decide to locally override for that message the final decision for the DMARC policy. How does the ARC headers checks actually works ? Through the ARC header the final recipient need to be able to look at the value of the original Authentication-Results field as it was created by the first hop in the mail forwarding chain, and then checks the authenticity of all the intermediate steps that have been traveled through by the email from that first step up to the recipient point. So, the recipient will be able to validate the entire chain. Thus an intact ARC chain provides to the recipient the following information : it gives DKIM, DMARC and SPF results as they have been seen by the first hop in the mail forwarding chain. Then it provides signatures that show that these results were actually not tampered with someone else. Finally, signatures from participating intermediaries can be linked to their domain name to prove that they are actually what they pretend to be. However, this doesn't prevent intermediaries to alter the message content. The main point of ARC is that such alterations can be done only with attribution, with which we are able to identify who did such changes in the chain. ARC doesn't provide us a solution for any problem. For example ARC doesn't tell us how much trustable is the message sender or the intermediaries, it only provides us information of what they did with that message. It says nothing about the content of the message and what has been added by the intermediaries to the content itself. ARC introduces the following header fields :

- **ARC-Authentication-Results (AAR)** : it's an archived copy of the Authentication-Results as it has been seen by the first hop. Each intermediary will performs the checks locally. They will give rise to several AAR headers that will be relayed up to the final recipient. Given that they need to be ordered for the final recipient to correctly interpret their content and the evolution of these checks through out the full mail forwarding chain, the AAR header also include a special tag i that contains a sequence number. The only Authentication-Results field that will not be archived in AAR header will be the one that is calculated on the final hop, i.e. the one computed by the recipient.

- **ARC-Seal (AS)** : it's a header that includes a few tags that are needed to understand how ARC should work, and a DKIM style signature of any preceding headers. It's built like an incremental seal that provide a proof of the fact that all the ARC headers including AMS and AAR headers that are included in a message have been correctly

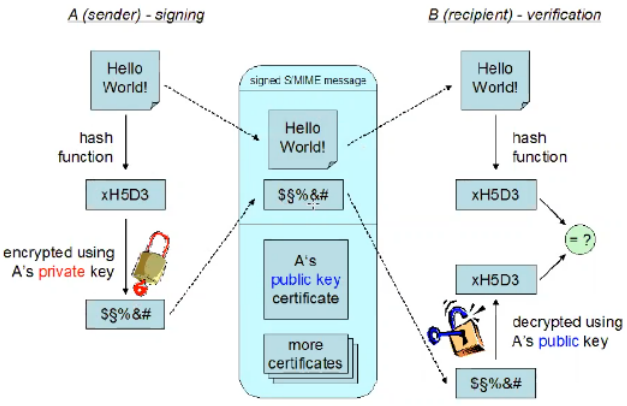
created and have not been altered during the travel of the message through out the forwarding chain.

- **ARC-Message-Signature (AMS)** : it's a modified DKIM signature.

The order of insertion is that first we have the AAR which are created with a simple copy of the Authentication-Results field, then the AMS and finally the AS. To summarize the ARC benefits for sender/intermediary are the following : it allows intermediaries to continue and/or resume traditional From semantics, message modifications, allows more senders to adopt strict DMARC policies, and improves overall deliverability. While the ARC benefits for the receiver are the following : less stress for enforcing DMARC policies, allows more mailbox providers to publish strict DMARC policies on their customer-facing domains and we have more data for reputation systems.

1.9 S/MIME

The authors of **S/MIME** protocol took all the approaches defined in PGP for signing and encrypting emails, and simply integrated them with a standard digital certificate based approach to trust. S/MIME is a widely used protocol for sending and receiving emails in a secure fashion, and its name comes from the fact that it somewhat extend the standard MIME protocol for encapsulating different kind of content in a single document, and it adds MIME types that are used on purpose to handle security mail content. Furthermore, it also adds some indication on how it implements email integrity and confidentiality. We can see that email signatures are handled exactly as in PGP



The original message is passed through a hash function in order to provide a digest, which in turn it's signed using the sender private key. Then the original message and the signature are embedded in a S/MIME message. Differently from PGP, the message typically includes the public certificate of the sender and may possibly include other certificates that are needed to correctly enforce the trust upon this public key certificate. On the recipient side, we have the classic approach used in PGP, i.e. the recipient computes the digest of the original message, decrypt the signature using the sender public key extracted from the sender public key certificate, and checks if the result matches the digest previously computed. When we move to encryption, it's performed exactly in the same functional way, but using X.509 standard certificates. A random session key is chosen on sender side, it's encrypted using the recipient public key and then the original message is encrypted using that session key. Both the encrypted message and encrypted session key are encapsulated in a S/MIME message plus the sender public key certificate. Then on the recipient side, the opposite approach is used to decrypt the message as in PGP but for the presence of certificates. The real strong difference between PGP and S/MIME lies in how the identity of sender and recipient is handled by the protocol.

1.10 How PEC works

A possible approach to implement a mechanism on top of the email service to include beyond security functionalities also tracing functionalities that allows to trace what happens to the emails and make these traces visible both to the sender and receiver goes under the name of **Posta Elettronica Certificata (PEC)**. It works as follow : the sender wants to send an email to the receiver but he wants also to have a confirmation about the steps performed to manage this email in its travel from himself to the receiver. It's mainly based on the usage of S/MIME, plus a set of procedures that are needed to create certified timestamps about what is happening to the email. In particular, the sender creates an email and ask its service provider to deliver such email to the receiver. The provider at this point only enforce the authentication from the sender, i.e. it checks if the sender is actually what he's pretending to be and the content of the headers in the email are coherent with the digital identity of the sender. Once all these checks are ok, the provider sends back to the sender a certificate, which contains information about such checks and that the email has been taken in custody by itself. Next, the provider will take the content of the email and encapsulate that email in an envelope that is signed by itself. The envelope is then sent from

the sender provider to the receiver provider through a reception point. The receiver provider checks if the signatures in the envelope are correct, and if it doesn't see any problem with the email, it will take into custody the email, and will send to the original sender a new certificate that attest the fact that he received the email from the sender provider with a specific timestamp and that he's now going to deliver the message to the receiver inbox. At this point, the receiver provider access the receiver inbox and place the message in it. When this happens, a new message will be sent to the sender saying "now the message is in the receiver inbox". In this way we have an email that is signed with a signed timestamp, and they are legally usable to attest the fact that an email was correctly delivered to a given recipient.

2 Introduction to Web Security

In the recent 15 years there was a strong increase in attacks that targeted multi-tier applications deployed on the Web. This increase follows the trend of the growth of what we call Web 2.0 applications, where the content is adapted to the specific needs of who is reading that and where web sites move from a situation in which they were mostly constituted by static content to a new reality where they represent full fledged applications. Naturally, protecting these kind of systems is way more complex than protecting a single standalone application on a computer, because simply there are way more levels of freedom that make these applications more dynamic, but on the other side make the job of security operators more complex. Notice that in this kind of scenario the critical components that needs to be protected are typically placed on the server side (DBMS server, authentication server, etc).

2.1 OWASP

This topic became so pervasive during the last 15 years that a few initiative arose trying to provide a comprehensive view of the problem and suggesting meaningful and still simple practical solutions. Among the various projects, one of the most interesting is called **Open Web Application Security Project (OWASP)**. It provides some best practices to be adapted in order to make systems more secure. The basic principles suggested by OWASP for securing complex web application are the following :

- Apply **defense in depth** : it refers to the fact that the best practices linked to border-based defenses are still good, as long as we define fine grained borders also within our premises. In other words, apply defense in depth means applying layered security both horizontally (working for different abstraction levels in our organization) and vertically for functions. This layered security allows us to create a lot more borders that we can control and enforce. Its main advantage is that if some adversary manage to overcome some of our defense measures at one of our borders, then there is quite a large probability that he will be able to gain an advantage point in our premises that is still not enough for him to reach its final intended goal. We should not rely on an unique defense.

- Use a **positive security model** : typically it's depicted as a choice between two opposite sides using a negative security model or a pos-

itive security model. This latter one is historically the one that need to be used and it's based on the usage of a whitelist. The idea is that, whenever we apply this approach, we need to deny everything and then making exceptions by whitelisting them.

- **Fail securily** : when a system for some reason fails, then is important to guarantee that it fails securily, i.e. errors are handled such that they do not creates security issues. From this point of view we can recognize two cases :
  - **type 1 errors** : they are errors that happens during the processing of a security control. Obviously they should disallow the operation.
  - **type 2 errors** : they are errors that are exceptions in code that are not part of a security control, but given how this code is designed, they may affect the way a security control is performed.
- Run with **least privileges** : with this approach we give to accounts the minimum amount of privileges that is necessary to perform a given operation. Of course, these privileges should last just for the time needed to perform such operation.
- Avoid **security by obscurity** : it's a strategy where the idea is that if you want to encode things what you should keep secret is just the key used to encode, but the algorithm should be free, open and known to everyone, because it's not by hiding things that you will make those things secure.
- Keep security **simple** : complexity could give us a false sense of security, but actually it's something that works against us. The more complex is a security system the easier it will be for it to contain errors, bugs, misconfiguration or bad design decisions.
- **Detect** intrusions : it's fundamental to log security-relevant events and to ensure that they are periodically monitored, in order to act as soon as possible to limit and mitigate the intrusions impact.
- **Don't trust infrastructure** : this means that understanding our infrastructure in details is fundamental to avoid errors and pitfalls that may give arise to incidents.

- **Don't trust services** : typically most applications are constituted by a set of open source libraries that works in a coordinated manner, but we cannot expect that they are secure.

- Establish **secure defaults** : we need to make sure that, even for people that are unaware of security pitfalls and threats, using our assets is something that can be done in a secure fashion. We may possibly give the opportunity to make informed choices on how secure is the usage of our assets and resources.

These practices works for a wide variety of potential risks on server side, client side, but also for ones that comes from the network.

2.1.1 Server side risks

They are typically the most effective ones and they are related to different objectives that adversary may have when they plan how to attack the servers we are running. Their goals are typically the following : **data stealing** (i.e. data breaches), **remote command execution** where the attacker is trying to deploy and execute on the target machine, code that is written by himself for various purposes, collecting further data for setting up more complex intrusions or attack the machine through a DOS to interrupt the services provided by such machine. We are dealing with the security of applications that offers services to end users, and it's important that these services are offered in a safe way such that the users will be able to access functionalities offered by services as they have been designed by the system designers. An attacker to subvert the functioning of the server tries to modify the application itself either to hide behavior that not directly visible to the users like steal their information or possibly to push the users to do some actions that would never do in a normal scenario. For this reason the Web server that run the application plays a fundamental role. In particular, it's very important that their configuration is appropriately managed by system administrator to make sure that content and applications are executed in a safe environment. From this point of view there are few elements that we need to take into account :

- **document root** : it's the folder where the web server will look for content to be served against users requests. Typically inside this folder we will find documents and files that represent the starting point to serve the content towards the users. Clearly the document root is another delicate point in the configuration, since its content needs to

be reachable from users with privileges such that to avoid that they can possibly execute code that they provide inside this directory.

- **server root** : it's the directory where the files and executables needed to correctly execute the web server are actually located. It typically contains few scripts, logs and configuration files. It's fundamental to protect this directory, since for example the logs files may contain sensitive information such as session key, and an attacker may try to steal it in order to replicate a valid session. Furthermore, this directory has a lot of space and subdirectory that can be accessed with different level of rights in order to correctly execute the server. However, an attacker may leverage such space to inject its own executable and make it run when the server starts, and he could leverage it for example to receive commands from a remote server.

Most of these problems boils down setting up these directories with the correct file permissions wrt the configuration of the web server. In particular, we need to take into account that document root will contains mainly documents that just need to be read and that server root will contains files that needs to be executed or documents that shouldn't be accessible from normal users. A common approach used to secure directory and user access rights is by defining specific user name and group for managing the access rights to these two directories (e.g. www & wwwgroup). Typically, the server root will be the home directory of www user. Instead, the wwwgroup is used to make sure content editors will be able to access directories with the correct rights to write new documents inside the document root and its subfolders. For the server root typically only the www user should have read/write/execute privileges everywhere within such folder and its subfolders. Users part of the wwwgroup may have read and execute access everywhere, but write permissions only on content that needs to be updated. Notice that the web server should run with a user called *nobody*, which has minimal access rights on the target system. This means that clients accessing the web server will have exactly those rights. For this reason, it's important that the document root will give permissions for reading and executing to other users different from www user and the ones in wwwgroup. In some cases, we may want to implement through the web server selective access to the content of the document root depending on the browser's IP or authentication. However giving permissions to all users to be able to read those documents is not a good idea, because all local users will be allowed to access that content. In this case, we may possibly reconfigure the server to run with a different user from *nobody*, provided to that user minimal privileges

and belongs to the wwwgroup. In that case, we need to make sure that such group has a restricted access with write rights to specific sections where we know that the users will not be able to do any damage. If for some reason, the web server runs with a user different from *nobody*, we need to make sure that no log directory is writable for that user id, otherwise it may possibly be used by an attacker to write code to be executed or to gain access to protected data like the *passwd* file, by simply creating a symbolic link within the log directory and then read that symbolic link. In the typical scenario, the web server is started as a daemon with user id root, and this is needed to allow the server to stay listening on reserved port such as 80 or 443, and to write log files in a protected place. Naturally, if someone connects to the web server, this latter accept the connection and spawn a new child process that will be executed with limited privileges (typically with user *nobody*). This is a robust scenario because it allows the web server to have full control of the machine for correctly serving the incoming requests and make sure that such requests can be handled by low privileges processes such that if any of these processes is subverted the impact of the compromisation is as small as possible. If we allow to run the child processes with root privileges, then everything can be accessed by the users and possibly the attacker can use the vulnerabilities in the code executed by the web server to make damage like only an administrator can possibly do. Another scenario, is where not even the parent process has root privileges. However, in such case the web server won't be able to open port 80 and neither other well-known ports.

There are some optional services that may be offered by web server and that represent another source of concern of security risks. In particular, most web servers includes the following services :

- **Directory listing** : it's a service that allows the users to get a directory listing. This is a strong problem, because we may end up stuff in document root that represent backup files, temporary directories and files. Notice that just disabling this service, doesn't prevent attackers from accessing these files.

- **Symbolic link following** : it's a service that allows requests to follow symbolic link. It represent a possible security risk, because if in the document root is present a symbolic link to an external file, then a client that request to access to such link will read the content of the linked file. A possible solution is to use some configuration opportunities provided by the web service (e.g. aliases) to extend the

document tree beyond the limits of the document root. In this way the configuration is more explicit wrt a symbolic link.

- **Server side include** : it's a simple way to include in static content some small snippets of dynamic data, that was provided by incorporating an external file or including in the html static content the result of the execution of an external process. In particular, among the various server side include directives there is the *exec* directive, that allows the execution of external scripts or shell commands, took the outcome and include it in the html file before serving it to the client. This is a common security risk because it allows, if not appropriately controlled, to make the attacker run more or less any potential script in the system and serve whichever content he wants. Today most of the functionalities that were formerly proposed through server side include are implemented by more complex full fledged server side scripting techniques based on various languages such as PHP, Python, etc.

- **User-maintained dirs** : it's a service that allows users to automatically add to document root personal portions of file system. This is potentially a strong breach in security, because in the end server administrators typically are not able to fully control what users actually do.

There are also some cases where content creator are given the possibility to access some sections of the document root through another service like a ftp server, that was used to update the content of the web server. The problem is that now we need to guarantee a coherent configuration of both servers in order to avoid potential vulnerabilities to arise from their possible connections. An approach that is sometimes used to enforce a bit more security on web servers is to use *chroot* command, which creates a limited execution environment with root the provided path. The problem with this kind of approach is that processes will only see things that are visible through the new root. In this environment we can avoid to include any interpreters or sensitive data that can potentially be used by an attacker to execute code or steal information. Another important approach that is used to secure the environment that is around the web server, is to isolate the web server wrt other potentially sensitive assets like hosts and networks that need to be better protected. This approach stems from the idea that the web server is something that needs to be accessed by anonymous users. While other parts of our networks don't have this kind of requirement. This approach is realized in such a way that, even if the web server is compromised, the

attacker will not be able to get into the entire network. Typically the server is placed in an isolated network area called **Demilitarized zone (DMZ)**, which controls the access from the internal network to the web server, and forbids the access from the web server to the internal network. The implementation of this kind of network segmentation is done using an appropriate use of the firewall, which may give arise to the following approaches :

- **Dual homed host** : in this case the firewall is equipped with two independent network interfaces, one is attested on the Internet and the other one is attested on private network, the routing between two interfaces is disabled by default, and packets are allowed to flow between the Internet and the private network only by passing through an appropriate application that perform in general application level filtering. In this case we would place the web server between the dual homed host and the Internet, but the problem is that this kind of configuration is not flexible, because any exception requires an appropriate rule to be written at application level within the dual homed host to allow packet flows and with large networks this hardly scale.

- **Screened host** : this approach is based on a router called **screening router**, which is configured to block any connection from the Internet to a protected network, but any of these connection attempts is redirected to a special host called **bastion host**. It's the only host present in the protected network that allows to access and receive connections from the Internet, and in practice it acts like a proxy wrt any connection that goes from the Internet to the protected network and vice versa. Clearly, the bastion host needs to be strongly protected (hardened setup). It's often realized through secure operating system that are standard operating system configured to offer as less services as possible such that they will not be exploitable by attackers. Every activity happening on the bastion host is immediately logged and placed on a secure log service such to be continuously monitored for possible traces of incidents.

Another important aspect to take into account while protecting server is to continuously monitor what is happening in that server. The server typically exports information in a detailed log that contains all information about which kind of actions are performed. The idea is that if the server is compromised, the log can be modified by the attacker, in order to hide its activities. In this case it makes sense to equip the server with a host-based intrusion detection system, which is a software that runs together side by

side with the server on the same host, and continuously check and analyze what the server is doing. They are able to intercept some suspicious activity and in such case they may possible raise an alarm or take some remediation action such as terminating that specific process and so on so forth.

2.1.2 Client side risks

From the client side, most of the risks are related to the browser. They are a good target for attackers because they may possibly be used to run the user code to call whichever effect may be the interest of the attacker such as browser crash, damage the user system, etc. In particular, the violation of end user data is a compelling scenario, since these data can be used later to perform identity theft attacks. In general, the attacker may use the vulnerabilities present in the user browser to attack the user itself by compromising a system as soon as a system visit the website, possibly delivering on the machine code that represent malicious software like trojans or spywares. An example of trojan is the following : we visit a website which download and execute on the web browser some client side code that exploit a local vulnerability. An example of spyware is the following : it's a software that sell itself for performing some legal action on the system, but they are written in a tricky and unclear way in such a way to push the user to accept the proposed conditions, making him subject to some type of legal data stealing from its machine. Today there is a security policy that is enforced by all browsers called **Same origin**. The idea of such policy is the following : if you visit a website and you exchange some data with that website, another website cannot access that data. The term origin is defined using the domain name, application layer protocol and port number of html document running the script. Two resources are considered to be of the same origin if and only if all these values are exactly the same. Why this policy is so important ? Because it allows the browser to implement easily a strong security policy that somewhat create some clear and unavoidable barriers between distinct applications and avoid these applications to inadvertently exchange information.

Cookies They are simple pieces of information that a website can store on a visitor web browser. It's a way to overcome the main limitation of HTTP protocol, which is the fact that it's a stateless protocol, i.e. it doesn't maintain any conversational support. For this reason the standard builders decided to introduce the cookies as a tool that can be used for several distinct things, and among these things there is a way to implement a simple

mechanism to track user sessions. The idea is that every time a web browser request a web page, the web server will provide the content that has been requested, but it may also include some headers that includes cookies. These cookies are simply small bits of information. Typically to track session the web server will include a session ID (a pseudorandom number which is with high probability unique and that will uniquely represent that user every time the user interacts with that web server during that session).

Once the cookies are stored in the web browser, every time the browser will make a new request to the web server, it will send together with the request the cookies. On the web browser, cookies are stored locally in a dedicated storage and it will make sure that they are send back to the web server only when the user is visiting the web server for which that specific cookies has been designed. In particular, each cookie will contain a few components such as name and value (mandatory), plus other optional components such as expiry, path, domain and need for a secure connection. Typically, we can find two kinds of cookies :

- **session cookies** : they are removed when the browser quits (they do not have an expiry date)
- **persistent cookies** : they are cookies that has an expiry date which remains on the client until such date. Typically, they are not kept in memory, but they are also saved somewhere on disk. Their typical usage is for example to automatically authenticate again the user as long as he tries to log in in a near future.

Another interesting way to look at cookies is by differentiating between :

- **first-party cookies** : they are provided by a website that we target directly by making a request through our web browser.

- **third-party cookies** : they are injected by other websites that we don't have directly visited or we didn't do that explicitly. This may happen for example if we visit a website A, which provides the content we are requesting and possibly also includes some first party cookies. Then our browser analyzes the content that has been received by A and while rendering the web page it encounters an element in the page that requires further content be fetched by another website B. In order to do that the browser will open a new connection toward domain B and through HTTP request ask domain B to provide the resource that is needed to correct render the page. While providing that resource website B has the opportunity to send back to the browser a cookie (this is considered a third-party cookies). This is typically a trick that is widely used by marketing companies to implement large scale of tracking of users preferences. In fact, the tracking is obtained by embedding in a web page an advertisement that will keep track for example of how frequently the user will visit such web page. However, there is still a problem, because the same origin policy doesn't allow website A to exchange information with website B. In the standard scenario the website A will make a request to website B, but will not include the cookie that are stored by it, because this will violate the same origin policy. What B can possibly do, is to store on our browser information about our identity from its standpoint. The two identities on A and B are obviously not linked. However these two identities needs to be linked in some way in order to allow an exchange of tracking information between A and B for a particular user. How can they do that without violating the same origin policy ? They can use the so called **cookie syncing** approach. The idea is the following : when we ask A to provide a content at the beginning of a session and provide our user ID, A before serving us the content will provide us with a redirect HTTP response that will point us to website B. In this request A will include both its ID and our user ID as its known by A. Our web browser will bring us to B sending these information plus the cookie for such website. At this point B will match our local user ID with our remote user ID on website A. Now, for B is enough to redirect us again to A with an information that will tell website A, "ok, i've synched the ID, you can proceed with serving the original content". How much can we protect ourselves from this kind of information leakage ? The most extreme solution is to use an ad-hoc solution that completely anonymize our usage of the web (e.g. TOR). Another approach is at least protect our point of access to the web

by using a VPN. The usage of cookies for tracking user preferences is more and more endangered on one side by technical advancements like those introduced in the browsers and on the other side by regulators that tend to limit the freedom of tracking services to do whatever they want to profile users. However companies continuously look for new solutions to overcome these limitations. An example are the **super cookies**, which are information still stored in our computer by non-traditional methods. They are persistent, but the fact that they are not real cookies, they tend to be ignored by both regulations and technical means to get rid of them.

2.2 HTTP authentication

For accessing a lot of online services we need to authenticate ourselves. We have a physical identity, but then we act in a virtual world using a digital identity (a representation of ourselves in the digital world). When we asked to authenticate by a service, the service will challenge us to prove the fact that our physical identity can be represented by a specific digital identity while using that service. Authenticating means proving that there is a match between our physical identity as user that is accessing that service and the digital identity that we will use for accessing that service. There are several ways to authenticate users. We will start talking about **HTTP authentication**. It's a protocol performed by landing on a web page that requires authentication, where the web server will fires up the request for authentication and the browser will answer asking us to provide a digital identity in the form of a username and password that confirms the link between that digital identity and ourselves. The HTTP authentication is typically configured on the server side by configuring in the proper way the web server. It implements two different challenge-response protocols :

- **Basic authentication** : assume that we perform a request for a resource on a web server and the server is configured in such a way to require authentication in order to access that resource including in the response header the www-Authenticate header that will include at least the form of authentication required plus realm=string. The realm concept is used to contextualize the authentication to a specific security realm on both the client and server side. When the client browser receives the 401 error message since the client didn't performed any authentication yet, it will show to the user the classic popup asking for username and password. The username and password will be send back by the client towards the server replicating

again the same request, but adding the Authorization header that will contains the type of authentication used plus a string. This string contains an encoded version of the username and password (Base64 encoding of username:password string). When receiving this request with the header, the server will take the content of the username and password from such string, checks the credentials locally, and if everything is ok, it will return the status code 200 and serves the request, otherwise it will send back the 403 error code. We want to underline that this schema doesn't use any encryption method, so it's very easy to decode such string in order to retrieve the user credentials.

- **Digest authentication** : It's an authentication method slightly more secure than basic authentication. In this case, the server challenges the client using a nonce (together with www-Authentication and realm header). Then the browser will again ask the user to provide username and password, and it will send the information using again Authenticate header, with the username in plaintext, the nonce received from the server, the realm, the uri of the service that needs to be accessed, a response (which is a cryptographically hashed version of the username and password) and the algorithm used to compute the response. The client to compute the response uses a predefined cryptographically secure hashing function such as SHA256, and then simply first applies the hash to the string username:realm:password, then it applies the hash to method:digestURI and finally it will hash the string constituted by the first hash computed, followed by :nonce:, and the second hash computed. The server replicates the same approach and then will check coherence of the calculated hash with the response that was sent by the client.

Another type of authentication is the **form-based authentication**, which is a simple way to include all those authentication methods that rely on the user sending credentials through some kind of html form. Its basic idea is that the user is redirected to a specific web page on the web application that he's using. This web page provides the basic form where it request the user to provide its credentials. Naturally, the requests to this page needs to be protected through TLS. So the user filling the fields with the username and password, click the button to submit the credentials that are sent through the secure channel towards the web server, which it will validates the user identity. Exactly like the previous methods, it's designed to overcome the limitations imposed by the stateless nature of HTTP. By asking user to authenticate, the web application fulfill two different goals

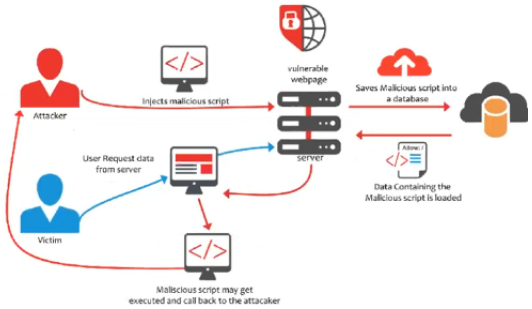
at the same time : on one side it creates a persistent session for the user, and on the other side it associate this session to a specific virtual identity. What the server typically does upon authenticating the user, it sets through a cookie a session ID for the authenticated user. This session ID needs to be protected, because it represents the authenticated user for the duration of its usage of the website. So, from this standpoint, the session ID represents the identity of a logged in user. This means that if someone is able to sniff and steal that session ID, he may possibly impersonate the user by sending new requests and independent from the ones of the real user, sending also the session ID. In this way the attacker is able to fully impersonate the user, unless the server implements some techniques to avoid this kind of attack. This attack goes under the name of **session hijacking** or **cookie hijacking**. So keeping the client-server channel secure by encrypting it by TLS is a fundamental requirement to implement form-based authentication and the subsequent secure session IDs secured through out the whole session. However, this is not the only way session hijacking can be performed. Indeed, if the attacker knows that the client-server channel is encrypted through TLS, he will look for alternative solutions to steal the session ID (attacking the server or the client). The security of the browser is typically linked to two aspects that are tightly coupled with the user that is actually using that browser (browser bad configuration or when the user give the permissions to the attacker to be attacked without understanding that). Fairly more frequent are attacks that target the server side. This kind of attacks typically target the front-end of the web application. They may have different purposes : target the web application to subvert its functioning, to steal data from the web application, attack specific users. Now, we want to talk about **unvalidated input** which is a typical software design flaw that give rise to different vulnerabilities that can be exploited with different kinds of attacks.

Unvalidated input Typically all web applications rely on the standard HTTP request-response protocol to provide their functionalities. If the client is an attacker he may try to subvert the application functioning by tampering the content of the data streams exchanged between clients and server. The server will take the data whenever it comes from and will possibly interpret them, or to use them to implements some functionalities. Now, if the web application blindly relies on the trust that it has towards its users, bad things may happen. Any data considered by the web application from the user input to be used for any reason, these data needs to be validated before

its usage. Unvalidated input from the user may possibly give rise to different kinds of attacks such as XSS, injection flaws and buffer overflow. Validating the input means implementing some functionalities that checks if the input provided by the user is fully compliant with the input that is expected by the application. There are several problems in implementing user input validation : client side or server side input validation ? However, the first approach is completely useless against a dedicated attacker, because he will simply skip it. So, the common case is that we should always implement input validation on server side. In order to proceed with input validation we need to use what is called *positive input validation*. It means that we as software developer should explicitly define the format and limits of the input that we expect. For example we can specify the minimum and maximum input length, the allowed character set, whether null is allowed, specific patterns and so on so forth. A kind of buffer overflow attack that comes from unvalidated input is the **Heartbleed attack** (for more information about that see here).

2.2.1 XSS

Another common kind of attack that comes from unvalidated input is **cross site scripting (XSS)**. XSS is a kind of attack that is strictly related to web applications. The typical case of a XSS attack is the following : there is a web application somewhere, that acquires data from a non-secure source (i.e. user input). The acquired data are sent to other users : they are the target of the attack. Once these users visit the compromised web application, they will receive data from the web application that contains input provided by malicious users. If this input contains client side code, this code is executed on the user browser.



An interesting web site to understand which are the most common attacks against web applications is **OWASP Top 10**. In particular, the information

for each attack are divided in three aspects :

- **Attack vectors** : this section contains information about how an adversary may possibly implement the attack
- **Security weakness** : it contains which kind of weaknesses enable this kind of attack and how we can possibly get rid of such weaknesses
- **Impacts** : it contains information about the potential impact of this kind of attack.

A typical way to perform a XSS attack is to embed some user input into a snippet of html code (e.g. inject a script in the $\langle a \rangle$ tag). In particular, there are three common variants of XSS attacks :

- **Stored** : it's a situation where the attack script is already stored somewhere on the website. The attacker uses a vulnerability present in the website, for example a form that doesn't correctly validate the input, to store in a database a malicious script. Then the content of the database is used to serve content to other users, and these users may end up running the malicious script.
- **Reflected** : it's a situation in which the user is pushed by the attacker in making a request to the vulnerable web server, that will send back the attack to the user itself. So the source code of the script is not stored in the server, but is the attacker that through some way (e.g. phishing email) push the user in sending the attack script to the server, and the server reflects back the attack script to the user, that is then compromised.
- **DOM-based** : We may have a vulnerable web page that includes a script that is provided by the web application itself. Some of these scripts may do some use of information that are provided by the user itself. If the input is not appropriately validated, then it can be used to perform a XSS attack. It's called DOM-based attack because the attack is not directly served by the vulnerable web application, but the attack is actually served by the user browser itself that renders the code containing the attack only at the end of the execution of some legit script provided by the web application. A difference with the previous approaches is that the vulnerability is executed on the browser, not anymore on the server.

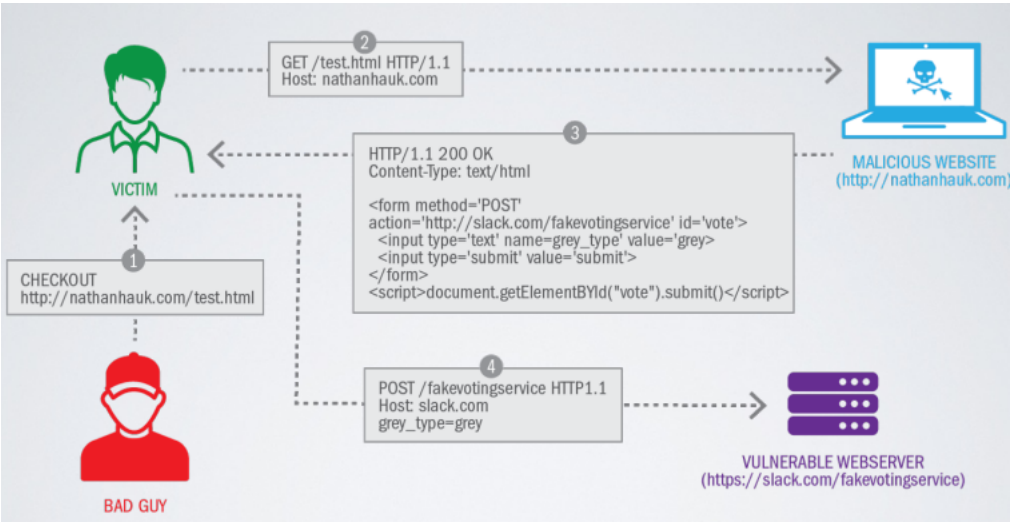
2.2.2 CSRF

Another kind of attack we want to talk about is **Cross Site Request Forgery (CSRF)**. It's similar to XSS attacks, the real difference is that we are not using scripts anymore, but rather we are targeting directly the web application using the user as a vector. The idea of CSRF is that the attacks works on the basis of a simple assumption : the web application is trusting whatever an authenticated user do. The problem is that the attacker with CSRF can push the user to send a request towards the vulnerable web application, without obviously the user being aware of that. The attacker proceeds as follows :

1. First of all the attacker observes how the users interact with the target web application. In particular, he may try to focus its attention on how the users provide data to the web application.

2. Next the attacker builds a fake request that mimics a valid user request. The only problem is that the attacker can't be considered trusted from the web application without a valid session ID.

3. To proceed the attacker lures an authenticated user to visit a website where he has included the form to be submitted (hidden wrt the user). Once the user visit the web page the request will start automatically. Thus, the data submission should be triggered silently and automatically as soon as the user visit that malicious web page.



How does the attacker perform a CSRF attack in practice ? Typically he needs to use a proxy to intercept a non malicious request, in order to understand how the request is performed, because he need to recreate it carefully. Then he setup the malicious web page that contains the CSRF attack with the hidden form and javascript to perform auto submit. Finally, the attacker send the phishing email to the user. The problem is that the target web application blindly trust the requests coming from authenticated users. The problem here is that the web server handles the request even if it's not actually following another request where the user asked the application to provide the form. The solution to this problem is to make sure that all form submissions requires some sort of random token to be submitted every time called **CSRF token**. This token is defined for each user session and be unique, secret and unpredictable. So every time the web application send a form to the user, it will include the CSRF token in the form as a hidden field such that every time the user click submit on the form, the value of the CSRF will return back to the web application and it will be able to check if the content of the token included in the form submission is equal to the token that was defined for that session. If the two are the same token, then it's a request that is coming as a consequence of serving a form to the user. Otherwise it's a potentially CSRF attack.

2.2.3 SQL injection

Injection flaws is a class of vulnerabilities that allows an attacker to circumvent the functionalities provided by the application and make the application execute code that was not in the original design of the application itself. This is the reason why they're called injection flaws, because they allows the attacker to inject its own code in the application and execute the code in the application context. The most common injection attack is called **SQL-injection**, which allows an attacker to inject code in the application in places that actually accepts user input, that will be executed by the interpreter behind the web application itself. In SQL injection attack typically we have that the application on the server makes use of a database back-end to store and retrieve data that is then presented to the user. How does SQL injection works ? The idea is that instead of providing the expected user input to the web application, we provide SQL code to be injected. In fact, the typical scenario, will dynamically creates a SQL query using some data provided externally that can possibly be altered by the attacker, without good validation of such data. Naturally, the target of this attack is always the server that serves the application and the attacker goal is to perform

operations that are typically not directly authorized. An important aspect is the fact that if we just consider the case where the attacker wants to inspect and extract data from the database, the problem is that the attacker is limited by how the output of the query is used to craft the interface that is then returned through the web response. In this case the attacker needs to use a different technique that is called **blind SQL injection**. In blind SQL injection the attacker doesn't see directly the result of the query, the only thing that he can see is a binary result (e.g. an error occurred or not). In this case the attacker may ask to the database true/false questions in order to extract 1 bit of information at a time. In order to be robust against SQL injection we need for sure to validate in a proper way the user input, then we can also use additional options such as **parameterized queries** or **stored procedures**. The former are predetermined queries with placeholders that are provided by the runtime environment where the application is actually running. The parameters provided by the user input are used to provide values to those placeholders. The latter are subroutines that are provided by the DBMS itself, which can validate input at server side.

2.2.4 Broken authentication

Authentication being a critical functionality of a web application is typically subject to several kinds of attacks. In particular, it may present vulnerabilities that if correctly exploited by an attacker may allows the attacker to impact the security of the application. Broken authentication methods are mainly based on problems linked to how password-based authentication is performed and how session security tokens are handled. When dealing with the authentication and the usage of reserved functionalities from users, typically we need to take into account two aspects : the authentication (link between physical and digital identity) and the second one is how do we authorize a specific user represented by a given digital identity to perform or deny the access to some specific functionalities. Password-based authentication has some well known and strong limitations that are mainly related to how password are chosen and managed by the users and website. In particular, attacks to such schema may be related to several aspects of password management. Notice that whatever we do for implementing a correct password management at the server side, the weakness can be on the user side, because most users continue to choose bad password for their digital identities. Predictable passwords are bad because they are an easy target for dictionary-based attacks, reused passwords are even worse, because allows the attacker to perform the so called **credential stuffing attack**, where

they have large databases of real password and then try them again and again on real accounts to check if anyone decided to reuse its own password. There are some best practices in how to push users to choose in the right way their passwords. The first practice is pushing people use password managers. Password managers as long as they are used with a local database of passwords, are good solutions for forcing people to use complex passwords because the passwords are generated randomly by the password manager and use different passwords for different services, since the password manager store all passwords for all services and the user needs to remember one single complex password to access the password wallet. The other approach that should be used is the usage of two factor authentication. Using two factor authentication means we challenge the user for at least two authentication methods. Regarding password management at server side we shouldn't store

- passwords as plaintext, because they are easily captured at server compromising
- encrypted password, because if the server is compromised it's likely that the encryption key is also compromised
- secrets (e.g. password digest) deriving from passwords, because digest of dictionary words are easily attacked.

Sessions are subject to attacks through several approaches such as XSS. In fact, in a reflected XSS scenario we trick the user in sending appropriately crafted javascript code to the web application that reflects it back and make the user browser contact an external service controlled by the attacker and pass to this service information about the session ID. The session ID at that point is linked to an authenticated user and can possibly be used by the attacker to impersonate the user in the vulnerable application. Notice that we can mitigate it in several ways : by setting up the session cookie as a *HttpOnly* cookie the browser will avoid from sending that data together with the other cookie towards the adversary controlled web site. Another approach is to make sure that, during a session, the session ID is linked to a specific IP from which the user was contacting the server. Another kind of attack related to sessions that stems from a wrong implementation of session management is the so called **session fixation attack**. The basic idea is that the vulnerable web site accepts any session identifier. The web application should be designed in a way that it will accept client requests bringing only a session ID that have been issued by the application itself

and that it's not expired. This check needs to be done for each single client request. Otherwise the attacker can send an email to the victim, tricking him in contacting the vulnerable web application with a request that already contains a session ID. In this case, if the victim trust the attacker, the user will be challenged for authentication on the vulnerable web application, he will issue its correct credentials and at this point the session ID that have been crafted by the attacker will be associated to a valid identity. To mitigate this kind of attack, the server needs to check that each single session ID that is created, is stored somewhere for its entire lifetime and during its lifetime it can be only associated to a specific user and after its expiry date the session ID should be deleted. Furthermore, when a user perform logs out from the web application its session ID should be deleted. In such a way, the attacker is not able to reuse session IDs associated to previously logged in users. Another attack is called **session side jacking**. It's based on the idea that the attacker may possibly setup an attack where he tries to sniff the content of requests between the user and web application. Some sites use TLS encryption for login pages to prevent attackers from seeing the password, but do not use encryption at all for the rest of the site, once the user is authenticated. However, the attacker may sniff the network traffic to intercept all the data that is submitted to the server, including the session cookie. This is one of the reason why today most applications tends to switch to a new model where everything is always served through secured channels. Notice that beside the usage of secure channels, session hijacking attacks can still happen if the user machine has been compromised through a malware, because in that case the malware may possibly tamper the web browser stealing the session ID from the browser memory space and sending it to the attacker blindly for the user.

2.3 CSP for mitigating XSS

We know that XSS attack is a consequence of the fact that there is a vulnerability on the server side, because typically user provided content is not correctly validated. Can we do something on the client side ? Yes, we can use **Content Security Policy (CSP)** to mitigate various types of attacks, including XSS. In practice, CSP is made up by a HTTP response header that instruct the browser on how to make use of the scripts and content that are loaded starting from the web page that is served. In particular, this will limit the kind and origin of scripts that can loaded. CSP is based on the idea that server administrators can through it reduce or possibly eliminate all the vectors through which XSS attacks are executed. In practice,

the server provides all content to client that perform the request including a content security policy through which a policy is defined (i.e. *Content-Security-Policy : policy*). In the policy what the administrators does is to specify a set of directives. Typically every policy should include a *default-src* token that identifies the default source that is accepted for content that should be loaded together with the web page (e.g. the *self* value means that, by default, content can be freely loaded by the browser from the domain itself from which the web page was loaded). We can also specify directives for handling different kind of resources such as *img-src* for source of images, *media-src* for source of medias and *script-src* for source of scripts, etc. This is effective because it reduces the options the attacker has to include in a target web page some external content. In addition to whitelisting domains, the policy can also provide two other ways to specify trusted resources :

- The CSP directive can specify a nonce and the same value must be used in the tag that loads a script. If these value doesn't match then the script will not be executed.
- The CSP directive can specify a hash of the contents of the trusted script. If the hash of the actual script doesn't match the value specified in the directive, then the script will not be executed.

2.4 Access control attacks

In general access control is a system implementing a methodology that enables some authority to control how users can access specific areas and resources that are made available by an application. In particular, through access control systems we can decide which specific user is allowed to access some specific resource. Notice that access control mechanisms are typically a critical defense mechanism for any applications. When an access control mechanism is defective, typically the attacker can take control of the full application. They are typically tight to the specific applications that are complex to implement and they are often a source of errors. We usually recognize two large families of access controls :

- **Vertical access control** : it allows different types of users to access different parts of the application functionalities. It's commonly used to enforce business policies like separation of duties and least privilege.
- **Horizontal access control** : it allows users to access some subset of resources from a wider range.

In general access control vulnerabilities takes the form of users able to access functionalities even if they are not authorized. We may have two main types of attacks :

- **Vertical privilege escalation** : when a user can perform functions that their assigned role doesn't permit them to do.
- **Horizontal privilege escalation** : when a user can view or modify resources to which he's not entitled.

In particular the access control main weaknesses are divided in the following categories :

- **Completely unprotected functionality** : it's a situation in which sensitive functionalities and resources are partially protected or not protected at all and they can be accessed by knowing the relevant URLs. So we need to take into account that URL can be guessed or brute-forced. Furthermore, links that are created by people accessing with correct credential, will appear in browser history and in the logs of web/proxy servers and they can possibly be read by other people. People can bookmark these URL or email them around.
- **Identifier-based functions** : it's a situation when the identifier that is needed to access a specific functionality is passed to some dynamic web page for example as a parameter. The only thing that the attacker needs to know to access that functionality is the page that provides this resource and the identifier of the resource itself.
- **Multistage functions** : in this case different items of data are captured from the user at each stage (e.g. sequence of forms). This data is strictly checked when first submitted and then is usually passed on to each subsequent stage, using hidden fields. Often developers think that if the user passes the first stage then a further check for that user is not needed since he has already passed the first stage. This is wrong, because the attacker will easily skip the intermediate stage guessing the information that is needed and passed from stage to stage and directly land at the final stage will all the needed information. Another point is that, it's a wrong assumption to think that people will blindly always follows the path that has been provided by the application designers, so that they will always start from the first stage and go down sequentially up to the last stage. The attacker will try all the possible combinations in order to check if by using other paths to access the

various stages he may possibly find a way to sidestep the authorization procedure.

- **Static files** : in this case the requests for protected resources are made directly to the static resources themselves, which are located within the web root of the server. At the end we get a static link to a specific resource that is completely unprotected. This means that if an authorized user is able to obtain that link, he will be able to reach the protected resource completely sidestepping all the authentication and authorization procedures. The problem with static resources is that they are not meant to be dynamic code executable by the web server. In some vulnerable applications there aren't effective access control mechanism that place a barrier before accessing that static resources.

2.4.1 Securing access controls

Unfortunately access controls mechanisms are often one of the most vulnerable points in a web application. This stems from several pitfalls that may together act to make our application less secure. First of all a lot of developers tends to implement access control, without properly thinking about the model they need to implement. They have some ignorance about which are the requirements for accessing sensible resources in their application and the lack of a proper knowledge about these requirements is typically a source of vulnerabilities. Furthermore, they typically also make flawed assumptions on the way users may access these resources by performing requests. Another problem is that web application developers often implement access control mechanisms on a piecemeal basis, simply adding code to individual pages. While implementing properly access control we need to take into account that : we can't trust any user-submitted parameters to signify access rights (e.g. admin = true), we should avoid to assume that users will access application pages in the intended sequence (so implement security mechanisms in depth), we should not trust the user not to tamper with any data that is transmitted via the client. If some data has been validated and is then transmitted through the client, we shouldn't rely upon the retransmitted data without revalidation. First of all access control is not something that we can design while we are implementing the application. We need to properly define, evaluate and document all the requirements linked to access control for every single resource that we want to protect. Naturally we need to take into account least privileges and separation of duties concept in the design phase. All access control decisions cannot be taken on the

basis on what the user submit, but only on the basis of the user session. It's a good idea to rely on a central component to check access controls. The advantages of this choice are the following : access control is simplified and easy to understand, the maintenance is more efficient and reliable, improves adaptability and it results in fewer mistakes and omissions. Once we have this central component, we need to make sure that every single user request is authorized by passing through this component. We need to include code that make sure that there aren't exceptions to the previous points. An effective approach from this standpoint, is to make sure that every single functionality is implemented without access control, but the access control is implemented in all pages by intercepting the requests, checking the rights to access such functionalities and only if the central component provides a positive answer then the functionality is executed. If we have extremely sensitive functionality, we can increase the security level by adding further checks such as restrict access by IP address to ensure that only users from a specific network range are able to access the functionality. Access control wrt static content can be implemented through a few methodologies. In the most simple case we can protect this content through basic HTTP authentication and then the user provides the credentials to access this resource. The other approach is to mask the location of these resources through a properly designed dynamic server side page. We link the server side page through which we pass a parameter, and this parameter is then mapped to the real resource that is only accessed by the dynamic web page that then reroutes the content to the user. This is a form of security by hiding but it's reasonably effective. So, resources still have a direct url that can be used to access them, but the point is that this url is never given to a client. Every time we have a web page that push the user to click on links that embeds as a parameter the identity of the resource that need to be access after proper access control, we should not rely on the content of such parameter because it can be easily tampered by an attacker. For extremely security critical application, we may think about implementing per-transaction authentication and authorization procedure. Every time we access the website as an authenticated user, if we ask to access a specifically critical functionality the system will double check our identity. Another important point, is that we need to log every access to sensitive data and functionalities because this will help any time we have an access control breach and possibly fix potential vulnerabilities. Finally, we need to use a layered access control to make sure that each single step is monitored and correctly authorized. This strongly mitigate possible pitfalls that arise from a monolithical design of our application. The nice point of using layer access control is that even

if we forget for example in a specific view of the UI to implement access control, even if the user sees a link that is not authorized to access, when he will click it, a lower layer access control mechanism will check and make sure that the user will not access that functionality if not authorized. For this reason, we need to take into account the possibility to adopt a standard multi-layered privilege model such as **Discretionary Access Control (DAC)**, **Role-Based Access Control (RBAC)**, etc. In general attackers will check every possible path and strategy to subvert the access control mechanism. For example in most applications, analyzing the logs we can see a lot of attempts to access directories with strange names that are not even present in the web application itself (typically the attackers uses automatic tools for doing this). The suggestion here is that when we have to design the access control we need to look everywhere, and check the absence of vulnerabilities, otherwise an attacker could leverage them and subvert the functioning of the access control itself.

2.5 Protect data in transit

We know that whenever we ask our users to use an ordinary HTTP connection to interact with our application, we are actually exposing the communication between our server and the users to some risk that arise from the fact that data is transmitted as plain text on the Internet. This makes the interaction between these two actors subject to MITM attacks. So, we should always consider the Internet as an adversarial environment for the simple reason that we have no control on its security. This means that usually all the links traveled by our data while coming and going between our server and clients are probably secure, but the problem is that we will never have an assurance about that. The best option is to assume that adversaries may control any link or router in the path between the two endpoints. To solve this well known problem, the standard solution is to setup an end-to-end secure channel based on TLS. So, this means that we will use still HTTP, but this time the data will travel through the secure channel and it will take the name of HTTPS. The guarantees provided by HTTPS are the following : all the data that travels through this channel will remain confidential (only the two endpoints can read the content of the data), no changes will happen to the data when traveling on this secure channel or if this happen it will be clearly visible from the receiving endpoint, and provides support for authentication whenever this is needed. There are still a few critical paths that we should be aware of such as protecting login procedures and sensitive transactions. The problem is that unprotected

functionalities may leak the session ID that is setup as a transient cookie after we login (MITM attacks for session side jacking). We understand that all my web pages will be served through HTTPS, but then each single web page include links to other resources that may be referenced through HTTP. In this case, our browser loads through HTTPS the web page content, and finds a reference to an external resource that is referenced through a link based on HTTP. The problem is exactly the same as before, if this resource is on our own web server, information about the session may possibly be leaked and may still be exposed to session side jacking attacks. The solution is to use TLS everywhere and to make sure all cookies are marked with the *Secure* attribute such that they are never sent by the browser through an insecure channel. Another less obvious anti-pattern lies in how we handle upgrades of a connection from non secure to secure. Typically, most applications actually redirect (301/302 response) users requests based oh HTTP to be served through HTTPS. Once the browser receives this response, it will simply redirect and make the same request to the same website, with the same domain, but using HTTPS. The problem is that this very simple interaction is vulnerable to the so called **SSL strip attack**. The idea of this attack is the following : the idea is to downgrade each secured connection through a non secure connection using a MITM attack. The first thing that the attacker does, is to setup a device to intercept connections from his victim. Then the attacker setup a proxy to proxy all connections that pass through that device. The idea is that as soon as someone tries to access a website there is a good probability that such user will do so by entering the name of the website on the browser and then the browser doing standard HTTP request to the server. This request that is in plain text, is intercepted by the attacker through its proxy and is passed to the real server that can possibly answer with 301/302 redirect response. Now the attacker avoid forwarding this response back to the user, but rather his own proxy follows the redirect and setup the HTTPS connection with the server, i.e. the proxy is pretending to play the role of the victim in front of the server. Once the proxy receives from the server the content through the secure channel, it takes that content, analyzes it and changes all the HTTPS links to HTTP ones. Once the translation to HTTP downgrading has been performed, this content is sent back to the client as a response to the original HTTP request. From this point on we will have that the victim has created a valid HTTP connection with the server. On the other side we have that the server has created a valid HTTPS connection with the adversary and the adversary in the middle from now on will mediate any possible subsequent request between the victim and the server and now he's

in the position to read everything that passes through this channel. In order to fill this gap that is present during the upgrade from HTTP to HTTPS, people started to think to the definition of a new protocol called **HTTP Strict Transport Security (HSTS)**. This is a web security that helps us to protect websites against SSL stripping and cookie hijacking attacks. The idea is that we can configure our server to send back with each request a header that will tell the browser that for accessing that website in the future should use only secure HTTPS connection, and never the insecure HTTP protocol. This header bring a sort of "switch" that configure for that web server a flag in the browser that will force it for future connections to that web server to only use HTTPS connections. Once the switch is received from the browser, it will automatically connect to such web site using only HTTPS connections. Naturally this close the gap, but if we receive this header at least once. However, if we decide to make a switch from HTTPS based connections to implement HSTS policy, we may easy encounter some problems at the beginning, because if we switch to full HSTS we may have some resources that stops to work. What is suggested is to perform this switch only for a given amount of time, by including in the header a TTL for the flag. Next, we can proceed with this approach by extending a little bit the flag duration until we make the switch permanent, i.e. the flag is valid for some years. This still leaves open just a small gap that is we need to receive this header at least once. Even if this reduces the temporal window availability to perform a SSL strip attack, to further mitigate this kind of attack the policy also includes a mechanism called **HSTS policy preloading**. The idea is that there is a website where we can register the domain name of our web application such that it will be included in a list that can be loaded by browsers before accessing any website. This list includes the list of websites that implements HSTS and that should never be contacted through an insecure connection. The idea here is that this preloaded list allows us to also skip the first insecure connection needed to receive the flag.

2.6 The cost of vulnerabilities

The problem is the following one : the vulnerabilities are bugs that can be exploited by an attacker to compromise the functionalities provided by the vulnerable application impacting the system's information assurance. A vulnerability is actually an intersection of three different elements : first of all the system that is vulnerable should present some flaw, then this flaw should be accessible from the attacker standpoint and he needs to have the capability to exploit it. Today vulnerabilities represent the major reasons

behind attacks. The usual vulnerability life cycle is the following :

- **Creation** : it represents the point in time where a vulnerability is created. This happens for example a developer write lines in a software, and these lines includes a bug that is exploitable.
- **Discovery** : it represents the point in time where someone discover the presence of a vulnerability in a software. A vulnerability can be disclosed by two kind of people : malicious users and benign users. The first case may give arise to the exploitation event.
- **Exploitation** : it represents the point in time where a malicious user is able to build a piece of software that represent a proof of concept about the exploitability of that vulnerability.
- **Disclosure** : it represents the point in time where the existence of the vulnerability became something of public domain. This disclosure event marks an important transition during the vulnerability life cycle, because any other event that happens before this event is constituted by non public information. When the disclosure event takes place, from that point on, the existence of the vulnerability is something public.
- **Patch** : it's the point in time where producers, once discovered that their software is vulnerable, plan for the production of a patch, i.e. a piece of software that modifies the vulnerable software such that the vulnerability is fixed or at least is made unfeasible to be exploited. Typically the production of a patch may require some time, but as soon as the patch is available, people will start to apply it to the software. The problem is that this event is purely hypothetical, i.e. this event may never happens in real life.
- **Software EoL** : it's the point in time when the software house decides to retire or abandon the software and we can assume that from this point on no one will use such application. For this reason, the life cycle of a vulnerability ends exactly when this event occurs.

Now the question is who pays for the presence of a vulnerability ? The cost depends on the time when the various events during the vulnerability life cycle actually take place. Notice that between the creation and the discovery of a vulnerability there is no cost. Then when the vulnerability is discovered the cost may possibly stand on two sides : on one side it will

stand on the shoulders of the vendor that by being aware of the presence of the vulnerability will start think about how to create a patch for that (the vendor pays in any case a cost even if he doesn't produce a patch, because several instances of its software are considered unsafe and its clients may possibly start to abandon the vendor's software), on the other side a cost may be paid by the black market where possibly the vulnerability will be sold. As soon as an exploitation is available and attacks will start, we will start to incur in an user cost.

3 Tor

The Internet was designed as a public network and everything we do on the web is tracked and logged in general. Nevertheless today we have a lot of tools that we use everyday and that scale perfectly to guarantee the security of our interactions with a service (e.g. encryption). However, these tools doesn't do anything for guaranteeing our anonymity. If we setup a HTTPS channel, and someone places itself in the middle trying to check the content of the channel, he will not be able to look at the content, but he will perfectly aware of the fact that we are trying to communicate with a specific website at a specific point in time. He will able also to track statistical information such as how many times we connect to such website, average session duration, etc. For anonymity we means that whatever we do on the web, we should not be identifiable as the source or the destination of that activity. It stems from two different aspects :

- **Unlinkability** of actions and identity : if someone observes what we are doing, he is not be able to understand what we are doing.
- **Unobservability** : even if an adversary is observing what we are doing, he has no way to understand which system or protocol we are using.

The attacker to break unlinkability requirement can possibly perform passive or active traffic analysis. The first one means trying to look at packet flow on a network link, and try to infer who is talking whom. Active traffic analysis where the attacker is able to actively inject data in the system and may use the effect of these network data to try to identify someone or some service. We also assume that, the attacker in order to perform its analysis is able to compromise some network nodes such as routers, firewalls, etc. The idea is that we can't trust the underlying network on top of which the Internet is based for anonymity. Actually we need to build something on top of the network that allows us to overcome all the threats paused by the previous threat model and still guarantee unlinkability between users while they communicate. There are several proposals dedicated for implementing anonymous services for end users, **The Onion Router (Tor)** is by far the most successful among all these projects. TOR is a service that we run on our device and other software can use to route traffic in anonymous way towards the desired destination. In general, TOR is distributed anonymous overlay network that allows participants to exchange information in a completely anonymous fashion. An interesting point is that, thanks to its architecture,

TOR allows its users also to access open standard web services. In this way the client is completely anonymized wrt the server. TOR offers another kind of functionality that goes under the name of **hidden services** that also implement the other side of anonymity. It allows users to access in an anonymous fashion services that are by themselves anonymous. In this case both the client and server identity are not revealed to the other endpoint of the channel.

3.1 Architecture

In Tor we have the following actors :

- **Client** : the user of the Tor network
- **Server** : the target TCP application such as web servers
- **Tor router** : the special proxy relays the application data
- **Directory server** : they are servers holding Tor router information.

How Tor works ? Assume that Alice is trying to contact Bob to obtain a service from him. To do that, the first thing that Alice needs to do is to create a protected, encrypted and anonymous link to Bob using the Tor network. In particular, she contact Dave, which represent one of the Tor directory server, to obtain a link to a certain number of Tor relays. A Tor relay is a Tor service running on the pc of someone on the web. At this point Alice will receive the identity of 3 relays (we assumed assumed such number for simplicity) and use them to create an encrypted link, where each of them represent an intermediate hop before contacting Bob. In other words, Alice is creating a virtual circuit within the Tor network, that will make her exchange data with Bob, but indirectly. Alice will be anonymous wrt Bob, because her packets will be indirectly received by Bob through the last Tor relay in the virtual circuit setup by Alice. So why the directory server provides Alice the identity of 3 Tor relays ? Because virtual circuits in Tor are made up of three relay nodes. The first node is called **guard relay**, a middle relay and an exit relay. The guard relay is the first Tor relay that is contacted directly by Alice to send data towards Bob. The exit relay is the last relay node in the virtual circuit, that has the role to interact directly with Bob. The middle relay is in charge to decouple the interactions between the guard relay and exit relay. Even within the Tor relay network, connections in a virtual circuit are performed through intermediate nodes. This approach is completely different from the standard

approach used by VPNs. Suppose Alice connect to the VPN using some protocol such as IPSec, and this means that we are setting up a virtual channel that is encrypted between us and the VPN server that is provided by our VPN provider. Next, the VPN server will immediately contact Bob (there will be a single indirection node between the two endpoints). This is a problem for the anonymity endpoint, because while Bob still sees packets coming from a VPN server and not directly from Alice, the VPN server itself has full visibility of what happens between Alice and Bob. Thus the unlinkability property that is needed for anonymity is not satisfied by VPN services. The usage of three intermediate relays in the construction of the Tor virtual private channel, is exactly what is needed by Tor to guarantee unlinkability. Notice that, a virtual circuit typically tend to last the whole session of an interaction between Alice and Bob. Thus, the traffic between Alice and Jane, will pass over a completely different and uncorrelated channel wrt the channel between Alice and Bob. This is needed to avoid the Tor network itself to be able to snoop upon Alice interactions with external services. The magic of unlinkability is provided within a virtual channel by a routing approach called **Onion Routing**. The basic concept is that Alice will send data to its endpoint using the virtual channel, but will encrypt data whose destination is Bob using a specific encryption approach called **Onion-like encryption**. In particular, to send data Alice encrypts data applying several layers of encryption that are setup such that each layer can be removed by a single onion relay. The data will first be encrypted using a symmetric key shared between Alice and the exit relay of the virtual circuit. Next, the data is encrypted again with a symmetric key shared between Alice and the middle relay of the circuit. Then, the data is encrypted again using a symmetric key shared between Alice and the guard relay.

Tor circuit setup First of all the client establish a symmetric key with the guard relay and to start the creation of a virtual circuit. In doing so the client exposes its identity to the guard relay. Thus the guard relay will act as an intermediate actor between the client and the middle relay in the virtual circuit. In particular, the client receives from the directory server together with the identity of the relays some information, like their IP address, but also their public key. At this point the client uses this public key to encrypt a session key that is sent through the guard relay towards the middle relay. The middle relay receives a request to setup the virtual circuit and the guard relay will include in such request also a virtual circuit identifier. Now, the middle relay will setup the data structures associated

to this new virtual circuit and will decrypt the received session key, and will use it to create a symmetric key that will be used to exchange data with the client. The symmetric key is encrypted with the session key sent by the client, and it will send back to the client passing through the guard relay. In this way, the client and the middle relay can setup their shared key in such a way that the identity of the client is never disclosed to the middle relay. The same approach is used to setup the symmetric key with the exit relay, but the requests will be related by the guard and middle relay towards the exit relay. At the end of this protocol, all three relays will share a symmetric key with the client that is know only by the client and the specific relay node. Once the virtual circuit is created, then it can be used to contact other external services using the symmetric keys for encryption. In order to deanonymize the connection made by a user towards a specific endpoint, an attacker should control all the three relays in the virtual circuit.

The design of Tor is made up of several components that collaborate to implement the functionalities that we have seen so far. At the first level we have the overlay network which is constituted by virtual routers that works at user level, and that interconnect and communicate with other routers such to realize a network on top of another network. It's important to known that onion routers are implemented through software that normal people execute on their own computers or servers and it's made up to work with standard user privileges. Then we have the onion proxy, which is the software needed to be installed by users in order to connect to the Tor overlay network and make use of its functionalities. Tor for implementing at network level the way routers and proxy exchange information is by using TCP with TLS in order to avoid Tor traffic to be easily filtered on firewalls (in this way Tor hides that we are using it). Another characteristics of the network data flow generated by Tor is that all data sent by Tor is sent in fixed size cells in order to reducing the amount of information that can be get by looking at the size of the packets. However, the packet frequency is not always the same, since the exit relay of the virtual circuit adds some noise to the frequency of packets that are sent toward the endpoint. Other functionalities that are enforced by Tor design are the following : all packets are checked for integrity, there is a mechanism called **link throttling** to avoid congestion on links (especially for links that interconnects onion routers). There is also a form of traffic throttling that is applied at the level of virtual circuit; if in the virtual circuit there are too much requests going in one direction without enough answers coming back from the endpoint of the connection, further requests will be throttled waiting for the response to come back. However,

the problem of having a public list of onion routers is that they can be all filtered out, avoiding the creation of a virtual circuit at all. To overcome this limitation the Tor project also keeps a sort of secret list of alternative onion routers that are called **onion bridges**. The directory servers doesn't known the onion bridges, since they are not part of the public list of onion routers. If we live in place where we cannot connect to Tor, because the connection to standard onion routers is filtered, we may ask the Tor project to provide us with a link with the identity of an onion bridge. The bridge acts as an intermediate point between the proxy and the guard router in the virtual circuit. The bridge will create virtual circuits on behalf of the proxy that is behind a firewall.

3.2 Hidden services

Hidden services are Internet services that are accessible only through Tor overlay network, they are not visible or usable through the standard Internet. The idea of making a service available through Tor is that the owner and administrators of those services have their identity anonymized. Anyone can connect to this website without knowing who is running that website. The original idea behind hidden services was to create something that was resistant to censorship. Setting up a hidden service is similar to setting up a virtual circuit for a client that wants to exchange data with an external service. In this case, it makes sense that both client and the service has their own virtual circuit, because in this case we want to ensure the anonymity for both of them. Now, we need to understand how Tor make sure that these virtual circuits have a point in common, that is an onion router where the two circuits collapse on the same point.

Hidden service setup Suppose that Bob wants to expose a hidden service. The first step to provide such service is that the proxy on Bob side selects a certain number (typically 3) of onion routers chosen at random. These onion routers are called **introduction points**. These introduction points are contacted by Bob by setting up three independent virtual circuits. Each introduction point doesn't know the identity of Bob. Then Bob locally creates a sort of service identity token that contains several information such service name, unique identifier of the service, list of introductions points and the service public key. Notice that the identity of a hidden service typically in Tor takes this form *service_name.onion*. This token is then published by Bob in a public database. This database is a single place where users can possibly query for information about available hidden services. If Alice

wants to access the hidden service, she queries the database for the information token about the service. Then she selects at random an onion router that will be called the **rendezvous point** for the connection between Alice and Bob. Now the problem is that, Alice needs a way to say to Bob in an anonymous manner, let's talk using that onion router. For this reason, Alice creates a connection request token, that is made up by setting up a piece of information that contains a cookie (a random number generated by Alice) and the identity of the rendezvous point selected by her. All these information are encrypted using the hidden service public key, which was contained in the information obtained by querying the database. Alice takes this encrypted request and creates a virtual circuit toward one of the three introduction points previously chosen by Bob (list of introduction points obtained again from the database). Next, she send to the chosen introduction point this request token, asking to it to route the request toward the hidden service. Once Bob obtains the request from Alice, he will decrypt the request using its private key. First of all he checks the identity of the rendezvous point. If for some reason Bob decide that some rendezvous points are not eligible for setting up connections, he will simply drop the request. Alice takes a TTL on the request, if it's not satisfied within a certain amount of time by Bob, she will try to make a new request selecting a new rendezvous point. If the rendezvous point chosen by Alice is ok for Bob, then Bob will create a virtual circuit towards the rendezvous point providing as information the cookie. At the same time also Alice creates a virtual circuit towards the rendezvous point, and place on it a connection token that contains the same cookie. This is the way the rendezvous point will be able to match two different virtual channels one coming from Alice, the other one coming from the hidden service, that both contains the same cookie. Once the rendezvous point sees the same cookie coming from the two virtual channels, it will connect them and act as a router between them. This means that all data that needs to go from Alice to the hidden service will pass through the rendezvous point. In particular, starting from Alice the data will reach the rendezvous point, and then proceed on the virtual circuit between this latter and the hidden service. By using two independent virtual circuits, we guarantee that the rendezvous point will not know anything about the identity of both Alice and Bob (the hidden service itself). At the same time we guarantee that Bob will not know nothing the identity of Alice and Alice will not know anything about the identity of Bob.

Since the database represent a single point of failure, it's implemented in Tor by the whole community of onion routers through what is called **dis-**

tributed hash table. It works exactly as a hash table, the only difference is that it's implemented through different machines, where each of them stores a subset of all the possible keys. Then it's implemented through a distributed protocol that enables us to access every key-value pair that is stored, by simply querying one node that makes up this distributed hash table, and the internal protocol will make sure that, by forwarding our request through several nodes, we will able to reach the node that actually keeps in memory that key-value pair that we are looking for. Furthermore, this kind of data structure implemented mechanisms that allow to replicate data such that if a node that stores a portion of data of the table fails, the data will be replicated on a different node. They also allow for a dynamic implementation of the hash table. Nodes that implement the distributed hash table can enter or leave the overlay network, and data will not be lost. When a node enters the overlay network a portion of existing data will be moved on this node. When a node leave the network the data that it handles will be moved to another node.

4 Privacy in the electronic society

Privacy is the ability of an individual or group to seclude themselves or information about themselves and thereby express themselves selectively (Wikipedia definition). However, privacy is a very complex concept to define, because there exist several interpretations of such concept. An issue that has been considered quite often in the past is called **Nothing to hide**. The basic idea is that, in the past privacy issue was essentially a difficulty coming from the government, in the sense that the government was interested in information about people. The idea is that if we have nothing to hide, we don't care about the fact that the government is interested in information about us. However, this position is not probably completely correct wrt what happened in many societies and especially what happens now in the digital society. The problem is that if we have nothing to hide, what do we have to fear ? The problem is that here there is a major concern about the role of governments, where they are interested in information about people also for purpose that are not known. The main issue is that privacy is not to protect the presumptively innocent from true but damaging information, but rather to protect the actually innocent from damaging conclusions drawn from misunderstood information. The nothing to hide concept has one weakness because focus on the information associated with state security. It doesn't consider issues such as :

- **Aggregating data** : it's a combination of small bits of innocuous data, that when aggregated reveals useful things.
- **Exclusion** : it's a situation when the people don't have the knowledge about how their information are used and cannot alter wrong data.
- **Secondary use** : it represent the scenario that data are collected for one specific purpose, but they could be used for other purposes.

The issue of privacy nowadays is more complicated, since we have multiple concepts to clear such as personhood, intimacy, secrecy, limited access to the self and control over the information. Naturally, this issue arise in several context and aspects of human life such as data collection by companies, different social contexts (family, social networks) and control of private information against the collection of private information. In our desire of privacy we can distinguish two aspects as human beings : the first one is that our desire of privacy is evolving depending on the situation, but on the other side we are really willing to give our privacy because we don't for an

immediate reward or we do not see the dangerous of privacy for us but we will see for others. In particular, privacy can be seen as it is constituted by four components :

- **Solitude** : it's a scenario where an individual is separated from the group and freed from the observation of other persons.
- **Intimacy** : in this case an individual is part of a small group.
- **Anonymity** : it's a situation where an individual is public but still seeks and find freedom from identification and surveillance.
- **Reserve** : this refers to the creation of a psychological barrier against unwanted intrusion holding back communication.

The idea is to limit access to the information, since there are laws that prohibit the collection and disclosure of such data, and technological tools that facilitates anonymous transactions in order to minimize the disclosure of information. Another important point is that we want have control about our information, but this is facilitated thanks to the technology that gives us informed consent, keeping track of and enforcing privacy preferences. In the market we have **privacy policies** that allows the consumers know about site's privacy practices. So consumers can decide to accept or reject these policies, when to get in or get out and who to do business with. The major advantage of the presence of these policies is that they increases the consumer trust. However, they have also some problem such as they are often difficult to understand, hard to find, take a long time to read and changes without explicitly noticing their consumers. In particular, the cost of reading privacy policies is typically related to how difficult is reading them. In fact, typically they are constituted by several pages of very small characters. For example, if everyone read the privacy policy for each site they visited once per year the estimated cost in hours is about 244 hours per year, which is equivalent to approximately $3534 per year. What the companies typically does with our information is not make them public, so our privacy is preserved, but they will use them to send specific advertisements to us.

5 Introduction to Bitcoin

When we need to buy something on the web typically we use the SSL protocol and a credit card. This approach is very simple, because doesn't need any specialized software, it's compliant with the credit card mechanism and it's one of the most used method for paying in the web. However, using this approach we reveal our information to potential malicious sellers and these increases the number of possible disputes. Furthermore, this approach is an expensive method for the shop that sells the the item of interest of the user and for sure the transaction for buy such item is not anonymous but is correctly tracked by the credit card company. For this reason, many credit card companies proposes a mechanism called **Secure Electronic Transactions (SET)**. We want to underline that this system never became operative for several reasons : all users must have a certificate for guaranteeing their payments and was too complex to put in place from the requirements standpoint both for shops and users. Few years later Satoshi Nakamoto developed **Bitcoin (BTC)**, which is a digital coin that beyond the money characteristics has the two following additional properties : it's decentralized (absence of a central authority) and it's immune to sovereign censorships. We can see Bitcoin as a combination of a currency, a payment system and a collection of algorithms and software implementation. The primary goal of the Bitcoin inventor was to make a system with low transaction costs and anonymous. In march 2017 one bitcoin is worth about $1141. There are slightly more 15 million bitcoins, but the algorithm has been implemented in such a way that won't be no more than 21 million bitcoins if the protocol doesn't changes. Now the question is how can we use Bitcoin ? First of all we need to visit *bitcoin.org* website, and we create a wallet which has associated an ID. Then using a proper software for managing our wallet, it will create public/private key pairs for us as needed. For each pair, there is a corresponding bitcoin address, which is a 160-bit hash of the public key. Then we send the bitcoins to the recipient address and we sign the transaction with our private key, so that the recipient can check our identity. Notice that a transaction might also include a small transaction fee. Bitcoin uses the following cryptographic tools :

- **Hash functions** : its input can be any string of any size and produces in an efficient way a fixed size output. In particular, for a hash function we requires the following properties :
  - **Collision-resistance** : a hash function H is said to be collision resistance if it's infeasible to find two values x and y with $x \neq y$

s. t. $H(x) = H(y)$.

- **Hiding** : a hash function H is hiding if, when a value r is chosen from a probability distribution that has high min-entropy, given $H(r \mid x)$ it's infeasible to find x.
- **Puzzle friendliness** : a hash function H is said to be puzzle-friendly if for every possible n-bit output value y and for k chosen from a distribution with high min-entropy, then it infeasible to find x such that $H(k \mid x) = y$ in time significantly less than 2^n.

- **Digital signatures** :
- **Public key** :

Bitcoin is based on the concept of **search puzzle**. A search puzzle consist of a hash function H, a value id with represent the puzzle-ID chosen from a high min-entropy distribution and a target set Y. A solution of the search puzzle is a value x such that $H(id \mid x) = Y$. If a search puzzle is puzzle-friendly then there is no solving strategy for this puzzle which is much better than just trying random values of x. Now, we want to understand why designing a electronic cash is so hard. The first issue that we could face goes under the name of **double spending**. Suppose Alice that has X bitcoins and sends a message to Bob says "ok, transfer X bitcoins to Bob". Next, Alice at the same time sends also a message to Charlie to transfer to him the same amount of bitcoins. In other words Alice is trying to double spending its X bitcoins. Now, if there is a bank for handling this problem is very easy, but at this point we are at the starting point again because the bank knows everything and it's working for us and should be paid. A possible solution to avoid this problem is to maintain a block chain as a public ledger of all transactions. So when Alice wants to adds a block to this ledger for sending X bitcoins to Bob, the operation is permitted. Whereas, when Alice in a future time point, will try to add a block correspondent to the transactions towards Charlie, the operation will not be allowed, because there is a check that notice that Alice is not the owner of such amount of bitcoins anymore. Typically in bitcoin, each block contains several transactions and a new one is created every $\sim$ 10 minutes. Naturally each block should be coherent, i.e. coherence means that two transactions for the same amount of bitcoins from the same user should belongs to the same block, otherwise the Bitcoin community will easily notice that that user is trying to double its bitcoins. So the idea is to make this public ledger as the central authority for managing and checking all the transactions. The block chain used by Bitcoin is a

sequence of blocks connected by hash function. The idea is that the last block contains data (transactions done in the last 10 minutes) plus the hash of the previous block. Naturally, the previous block contains data plus the hash of the previous block as well. The idea of using a block chain for managing transactions is that if an adversary modifies data anywhere in the block chain, it will result in the next hash pointer being incorrect and we are going to detect it. If we store the head of the list and even if the adversary modifies all the pointers in order to be consistent with the modified data, the head pointer will be incorrect, and we will detect the tampering. However, the usage of a block chain doesn't guarantee us the truth, but preserve truth and lies from later alterations, allowing to securily analyze them and be more confident in uncovering the lies. We know that a user in the Bitcoin network are characterized by a public key, which digest is called **address** in Bitcoin jargon. Now, we want to see two attempts about the issue of double spending :

Goofy coin The first rule is that a designated entity called Goofy, can create new coins whenever he wants and these newly created coins belong to him. To create a coin, Goofy generates a unique coin ID that he's never generated before to be assigned to the coins previously created. At this point, he computes the digital signature for such coins using its secret key, so that everybody can verify that the coin contains a valid signature or not. Now, suppose Goofy wants to transfer a coin that he created to Alice. To do this he creates a new statement that say "Pay X coins to Alice", where X represent the amount of coins to be transferred to Alice. We recall that identities are just public keys, so Alice refers to Alice's public key. Finally, Goofy signs the string representing the statement. Everybody in the network can verify the transaction signature using the Goofy public key associated with the private key used for signing the transaction itself, and if it's valid then we can say that an amount of X coins has been transferred from a person to another person. At the end, everybody knows that such amount of coins are owned by Alice and not anymore by Goofy. However, this approach doesn't avoid Alice from double spending its own coins.

Scrooge coin Another approach is to use a centralized entity named Scrooge that publishes the history of all transactions that have happened. To do this it uses a block chain digitally signed by itself. In such a way everyone can verify if :

- the coins are valid, which means that they were really created in pre-

vious transactions.

- the coins were not already consumed in some previous transaction, i.e. it's not double-spending.
- the total value of the coins that come out of this transaction is equal to the total value of the coins that went in. In other words, only Scrooge can create new value.
- the transaction is validly signed by the owners of all the consumed coins.

In this case we need to trust Scrooge about the transactions. For this reason, if all the previous conditions are met, then the transaction will be accepted by Scrooge. In particular, Scrooge write the transaction into the history by appending it to the block chain, so that everyone can see that a new transaction occurred. In this way we avoid the double spending issue, because it's only when Scrooge publishes the transaction that the other users can accept that the transaction has actually occurred. However there are two problems with this approach : there is one person that is responsible for everybody and that we delegate one person to be in charge of everything. In other words, we need to trust Scrooge that is honest, but this represent a weakness since it can be attacked.

Another system that is trustable like Scrooge coin but doesn't have a central authority is Bitcoin. However, due to the absence of a central authority, in such system is very hard to reach consensus. Indeed a distributed consensus protocol has the following two properties : it must terminate with all honest nodes in agreement on the value, and the value must have been generated by a honest node. Essentially, when Alice send a message to Bob for paying him, she broadcast the transaction to all Bitcoin nodes that comprises the peer-to-peer network. In Bitcoin network we have the following consensus requirements :

- **Many users broadcast transactions** : the nodes must agree on exactly which transactions where broadcast and the order in which these transactions happened.
- **Many ledgers** : at any given point, all the notes in the peer-to-peer network have a ledger consisting of a sequence of blocks, each containing a list of transactions, that they have reached consensus on.

- **Keep consistency** : all nodes agree on a single global ledger for the system.

When a block is proposed by a node with a ledger, all the other nodes executes some consensus protocol in order to check if such block is correct or not. To propose a block to the community we need to solve a search puzzle, which is hard to solve. If the block is correct (you solved the puzzle) the block is accepted and you will receive a reward for it. However, at the same time there may be two people that propose correct blocks, but at that point there is some consensus protocol that decides which of the two blocks needs to be accepted. The problem with this approach is that the network is not perfect and for this reason there could be latency and crashing nodes or malicious nodes that try to subvert the process. Bitcoin consensus overcome the main problems of the traditional consensus models using two tricks : nodes receives rewards to be honest and that it uses randomization guaranteeing us that the probability of reaching consensus is overwhelmingly high. The consensus protocol used by Bitcoin can be summarized in the following way : first of all new transactions are broadcast to all nodes. Next, each node collects new transactions into a block and in each round a random node broadcast its block. The other nodes accept such block if and only if all the transactions inside it are valid. In positive case, the nodes express the acceptance of the block by including its hash in the next block they create. If there are more possibilities the node chooses the longest chain. The Bitcoin network can be subject to the so called **sybil attack**. In such case, the attacker creates several copies of nodes which are called **sybil**, so that they look like as a lot of different participants, when instead they are all controlled by the same adversary. Now the question is can we steal Bitcoins ? The answer is no, because if a miner proposes the next block stealing other user's bitcoins, would require the miner to create a valid transaction that spends that coin, which in turn means to forge the owner's private key to sign the transaction, but this cannot be done if a secure digital signature scheme is used. Another possible problem is the DoS attack, which comes from the fact that an user dislike another user. Suppose that A doesn't like the user B. So A can decide to not include any transaction originated from B's address in any block that she proposes to get into the block chain, i.e. A is denying the service to B. This may seems a valid attack, but if B's transaction is excluded by the block proposed by A, he needs just to wait until a honest node gets the chance to propose a block and then his transaction will be inside that block. Now how does the double spending issue is avoided using Bitcoin ? Suppose Alice at the same time

issues two payments with the same amount of Bitcoins towards two different addresses. The idea is that we need to kill one of these two transactions in order to avoid Alice from double spending its Bitcoins. In the procedure used for reaching consensus the nodes try to extend the transaction they are aware. The accepted transaction is the one that will be the longest, because nodes might be rewarded only by extending the longer chain. This is the reason why the Bitcoin community typically ignores the shorter chains, because in case of extending a shorter chain they might not be rewarded at all. Since the block containing the short transaction is ignored by the network, it's called an **orphan block**. Now the question is how long we have to wait to be sure that there is no double spending ? The intuition is that if we wait for more or less one hour, since a confirmation happens every 10 minutes, we will expect to have around 6 confirmations. Indeed, the number of confirmations provides confidence in the probability of the next accepted transaction and we have an exponential decrease in the probability of having a double spending issue. A node for computing a new block is required to find a nonce such that when it concatenate the nonce, the hash of the previous block, and the list of transactions that comprise that block and take the hash of such string, then such hash should be a number that falls into a target space that is quite small in relation to the much larger output space of that hash function. In particular, Bitcoin defines this target space as any value falling below a certain target value :

$$H(nonce \mid\mid previous\_hash \mid\mid tx \mid\mid ... \mid\mid tx) < target.$$

Naturally, if the hash function chosen is good then accomplish this task is computationally hard (it requires computation power and time). This task refers to the fact of solving a hash puzzle. Obviously, they needs to be quite hard to compute. For example at the end of 2014 the difficulty level for solving a hash puzzle was about 10^{20} hashes per block. We clearly understand that the size of the target space is only $\frac{1}{10^{20}}$ of the size of the output space of the hash function. The parameter of the difficulty is tuned on the basis on how many nodes are trying to create a new block every certain number of blocks by the protocol. This implies that if there are few nodes we will decrease the difficulty. Vice versa in presence of several competing nodes we will increase the difficulty. To do this the nodes in the Bitcoin network automatically recalculate the target every 2016 blocks. In particular, they recalculate the target in such a way that the average time between successive blocks produced in the network is about 10 minutes. With such average time between blocks, the 2016 blocks can be processed in roughly two weeks,

i.e. the target is recalculated every two weeks. For this reason only particular nodes with sufficient computing power capabilities can compete in this block creation process. This process of repeatedly trying and solving these hash puzzles is known as **Bitcoin mining**, and we call the participant nodes **miners**. The crucial point is that if we consider that the majority of miners are honest and follow the protocol then many attacks on Bitcoin are infeasible. This is true because if the majority of miners are honest, the competition for proposing the next block will automatically ensure that there is at least a probability of $\frac{1}{2}$ that the next block to be proposed comes from a honest node. Now the question is how long it will take a miner to find a block ? The average time for a miner to propose a new block is given by the ratio of 10 minutes and the fraction of hash power that the miner has. For example, if the miner has the 0.1% of the total network hash power, find a block to propose every 10000 minutes (about a week). To verify a block it's sufficient to check the hash of the block and that all transactions are correct. So, it's trivial that a node has computed the proof of work correctly. The nonce must be published as a part of the block, so that the other nodes can trivially verify that the output is less than the target. This is the reason why we don't need any centralized authority that verifies that miners are doing their job correctly. Furthermore, when a block is proposed there is a special transaction within in called **coin-creation transaction**, so that if the user that proposed the block is the lucky person that solved the puzzle than such user will give the coin just created to himself. We need to underline that this is the only way to create new coins and this is done in exchange to the work done for maintaining and proposing new blocks. The original reward was 50, but it actually halves every 210000 blocks. Halving every 210000 blocks gives a geometric series, which gives us a finite sum of 21 million Bitcoins.

Nowadays proposing a new block is becoming more and more expensive. This implies that the reward that you have for a block might not be sufficient for paying all your expenses. The second issue is that the popularity of Bitcoin implies that there can be many requests to be processed an put in a block. Another Bitcoin requirement is that the size of a block cannot be very large. This implies that more or less we can have 200 transactions in a block. If there are more than 200 transaction and we want that our transaction is inside a block, we can insert in the transaction a small transaction fee to the miner that will include our transaction in the block. This is like paying a service offered by the miner, but it's not mandatory. In this way a miner that receives a transaction without a transaction fee and a transaction

with a transaction fee, has an incentive to put the second transaction in the next block that he will propose. Naturally the costs of miners are given by hardware cost plus the electricity cost, and they have incentive to mine if and only if cost < transaction fee + block reward. We will expect that in 2040 the block reward is 0, and at this point we will really need transaction fees in our to allow Bitcoin to survive. We already said that the nodes in the Bitcoin network are not all equals. In general, we can distinguish two kinds of nodes :

- **Full nodes** : they are nodes that fully verify all the rules of Bitcoin and download the entire block chain. This type of nodes doesn't propose new blocks, but stores and answers to queries from other nodes. The full nodes main goal is to enforce the Bitcoin consensus rules. They have the following advantages :
  - **Security** : running a full node is the only way we can use Bitcoin in a trustless way, because it will check that all the rules of the Bitcoin consensus protocol are satisfied.
  - **Privacy** : a full node is currently the most private way to use Bitcoin, with nobody else learning which Bitcoin addresses belongs to us.
- **Light nodes** : they are nodes that don't download the complete block chain, but downloads the block headers in order to validate the authenticity of the transactions. For this reason, this type of nodes is easy to maintain and run. Typically light nodes are served by full nodes to connect to the Bitcoin network. However, they have some limitations such as :
  - **Validation** : they don't validate the rules of Bitcoin. So if someone pays a light node with fake Bitcoins, then this latter will accept them and the user will be left out of its money.
  - **Privacy** : they typically send addresses to a trusted third party and receive the wallet balance and history. At that point the issue is that they are transferring all the information about the wallets that is more or less like a bank.

We understand that the light nodes doesn't check the Bitcoin consensus rules and for this reason can be tricked by everybody in the network, whereas full nodes check those rules and thus enforce the validity of the block chain itself. If there is a fake evolution of the block chain that contains for example

a double spend transaction, then we may have a bifurcation of the block chain. This implies that that we will not have anymore a single Bitcoin community, but two Bitcoin communities each with its own currency. A direct consequence of such scenario is a decrease in the trust of Bitcoin, which will push people to abandon it and thus the value of Bitcoin will decrease as well. In practice, miners are unlikely to attempt the creation of such fork because as long as full nodes are prevalent they would lose a lot of money. To recap, it's critical for Bitcoin survival that the great majority of the Bitcoin economy be backed by full nodes.

5.1 Bitcoin mechanics

The transactions are not indicating transfer but ownership of coins. The bad point of such approach is that if someone wants to check if a transaction is valid then he needs to have the history of the transactions about the sender in order to check that at the time instant when the transaction has been triggered the sender has a balance greater or equal than the amount to transfer. A transaction is constituted by three components :

- **Metadata** : they represent information about the transaction such as its size, the number of inputs and the number of outputs. There is also the hash of the entire transaction which is used as a unique ID for the transaction.
- **Inputs** : it's an array of inputs. Each input specifies a previous transaction (the hash of that transaction), the index of the previous transaction's outputs that being claimed and a signature.
- **Outputs** : it's an array of outputs. Each output has just two fields : a value and the sum of all the output values that has to be less or equal than to the sum of all the input values.

In particular, Bitcoin offers the following services :

- **Change addresses** : typically when an user issue a transaction where its balance is greater than the amount of Bitcoins in the transaction itself, then he also issues a transaction also towards himself, because Bitcoins are immutable, hence the entirety of a transaction output must be consumed by another transaction.
- **Efficient verification** : suppose that a new transaction is added to the public ledger. How easy is to check if it's valid ? We need to

look up the transaction output that the user referenced, that it has a value of 6.25 Bitcoins, and that it hasn't already been spent. To ensure it hasn't been spent, we need to scan the block chain between the referenced transaction and the latest block.

- **Consolidating funds** : suppose that Bob has received money in two different transactions. If Bob wants to spend all the received money he just have to create a transaction with the two inputs and one output, with the output address being one that he owns, i.e. it's a way for allowing Bob to consolidate these two transactions.
- **Joint payments** : it's a situation where a transaction has two or more inputs and a single output and it's signed by two or more people.
- **Escrow transactions** : suppose that Alice wants to buy some item from Bob and she don't want to pay until she is sure to receive the object. To proceed Alice creates a MULTISIG transaction that requires two of three people to sign in order to redeem the coins : Alice, Bob and a third party Judy. So if there is no problem Alice and Bob sign and Bob can then use the coins. Otherwise in case of dispute, they call Judy as a judge to settle the issue.
- **Green addresses** : suppose that Alice wants to pay Bob, but Bob is offline and thus cannot check if such transaction is actually there. For this reason, we introduce a third party to which Alice ask it to send the money to Bob. This third party takes the money from Alice and gives them to Bob. Naturally, this third party should be trusted also for checking double spending.

A Bitcoin block is structured in the following way : 4 bytes for the block size, 80 bytes for the block header, $1 - 9$ bytes for the transaction counter and a variable amount of bytes for all the transactions recorded in such block. Instead the block header is constituted by the following fields :

- **Version** (4 bytes) : it's a version number to track the protocol updates
- **Previous block hash** (32 bytes) : it's a reference to the hash of the previous block in the chain
- **Merkle root** (32 bytes) : it's the hash of the root of the merkle tree of the block's transactions
- **Timestamp** (4 bytes) : it represent the approximate creation time of the block

- **Difficulty** (4 bytes) : it's the proof-of-work algorithm difficulty target for the block
- **Nonce** (4 bytes) : it's a counter used for the proof-of-work algorithm.

There is also a scripting language built on purpose for Bitcoin. It's very small (contains only 256 no loop instructions). Clearly having only one transaction in a block is very slow because it will require a lot of work. So, the idea is to group several transactions per block. Now how can we check the validity of each single transaction ? A transaction doesn't know to which block it belongs. For this reason, Merkle trees are used to check if a transaction is in a block. A Merkle tree is a tree in which we have the data in its leaves, and in the internal nodes two hashes of the corresponding child nodes. In fact, a node to prove that a transaction is included in the block can simply produce a Merkle path that is only 4 32 bytes hashes long. Now, how can we publish a transaction ? We can use a simple flooding algorithm that simply broadcast the transaction to all the nodes that are peered with us. Each of these nodes executes a series of checks to determine the validity of the transaction and in positive case the nodes in turn sends it to all of their peer nodes. Furthermore, when a node receives a transaction, it put the transaction in a pool of received transactions that aren't on the block chain yet. The propagation time of a block with a size of $200KB$ takes less than 30 seconds to reach the 75% of the whole network. Now, we want to talk about an attack called **51-percent attack**. It's a scenario where an attacker controls the 51% or more of the mining power in the Bitcoin network. Even in this case, the attacker cannot steal coins from an existing address, because he needs to alter the sender signature but this would require him to break the cryptography. The attacker cannot neither suppress any user transactions, because a transaction is simply broadcast in the Bitcoin network and we are assuming that the attacker is not able to fully control the network. So, the transaction will reach the majority of nodes, that in turn, if the transaction is valid, will insert it in next block that they will propose. Finally, the attacker has a very low probability to be able to decrease the confidence of the Bitcoin network itself, because the cost of achieving a 51% majority really makes sense from a financial point of view.

5.2 The miner role

A miner has to follow a straightforward procedure in order to get a reward :

1. First of all the miner listen for transactions on the network and validate them by checking that signatures are correct and that the outputs being spent have not been spent before.

2. Next, he maintain the block chain and listen for new blocks. In particular, a miner before joining the network will ask to the other nodes all the historical blocks that are already part of the block chain. Next, he listen for new blocks that are being broadcast to the network. The miner must validate each block that receives by validating each transaction in the block and checking that the block contains a valid nonce.

3. Once the miner has an up-to-date copy of the block chain, he can begin building its own blocks. In order to do this, he group the transactions previously heard into a new block that extends the latest block he knows. Naturally the miner needs to make sure that each transaction included in its block is valid.

4. Then the miner needs to find a nonce that makes its own block valid. This is the most time and resources consuming step for him.

5. Once the miner have found a valid nonce, proposes a new valid block. However, the acceptance of such block is not guaranteed. So the miner needs to hope that other miners will accept its block and start mining in top of it, instead of some competitor's block.

6. If all miners accept the miner's block then he obtains the reward (in 2020 the reward is 6.25 BTC). In addition he collect the transactions fees of the transactions in its own block (today they represent only about 1% of block rewards).

A transaction doesn't include a transaction fee if it's less than 1000 bytes in size, all outputs are greater or equal to 0.01 BTC, or priority is large enough. Otherwise a fee is charged and it's about 0.0001 BTC for 1000 bytes. We know that a miner is rewarded only on the basis on its hash computational power, but the problem is that he could be unlucky and doesn't get the reward at all. For this reason, a miner could merge its hash power with the ones of other miners in order to create the so called **mining pool**, in order to share the mining process costs.

5.3 Getting a cryptocurrency

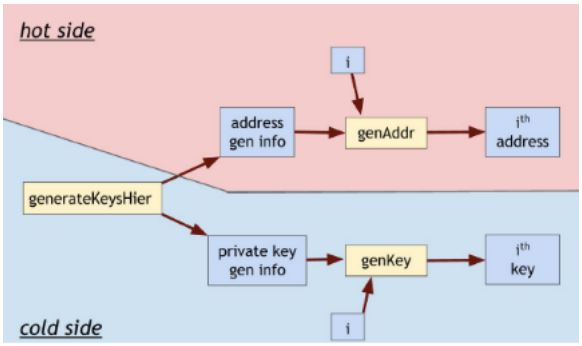
Now we want to discuss how we can go from the Bitcoin protocol to a real digital money. The idea of having a digital currency requires three aspects

: the security of the block chain, otherwise nobody will trust Bitcoin and nobody will use it, the health of the mining ecosystem, which means that if the miners don't get enough reward for their activities they won't do that for free and for this reason they may decide to not give computational power to maintain the system, and the value of the currency itself. When Bitcoin was first created, the previous three properties didn't hold. The critical point is that Bitcoin has not been proposed by a bank or a big company but by an unknown researcher known as Satoshi Nakamoto. Bitcoin is became so popular because the more people hear about it, the more they are going to get interested in mining and in turn this increases the confidence of people about the security of the block chain because there is now more mining activities going on, and so forth. Now the question is how can we maintain Bitcoins ? The simplest way is to store them on a local device such as a laptop. However, this requires to deal with Bitcoin secret keys. The problem is that if we lose the device or our file gets corrupted, our keys are lost and so are our coins. Furthermore, if somebody infects our device with a malware, then such person can copy our keys and then he can then send all our coins to himself. In general, we can store Bitcoins in the following two ways :

- **Hot storage** : it means that we store the Bitcoins on our computer, but this is very risky.
- **Cold storage** : this approach works in offline mode. In particular, it's locked away somewhere and it isn't connected to the Internet. This approach is safer and more secure wrt the previous approach, but of course it's not convenient.

Naturally to have separate hot and cold storage we need to have separate secret keys, otherwise the coins stored in the cold storage would be vulnerable if the hot storage is compromised. A common scenario is to move coins between these two storage, which implies that both sides needs to know the address of the other one. The problem is that in such scenario we would like to use a fresh cold address for the money transfer. This can be done by generating a big batch of addresses all at once and give them to the hot storage in such a way when a money transfer is issued the hot side uses them up one by one. A more effective solution is to use a **hierarchical wallet**, which allows the cold side to use an unbounded number of addresses and the hot side will know about these addresses for performing the money transfers. However, these addresses will be available only for a short one-time communication between the two sides (to do that we need to use some cryptography trick). In particular, given an initial address generation information, there

is a function that generates a sequence of public and private keys, which for any integer i it generates the $i-th$ address and the $i-th$ secret key in the sequence. Obviously, knowing the list of public key doesn't reveal anything about any secret key.



We understood that the ownership of BTC is given by knowing the keys and we must ensure that their retrieval is difficult. However, there are also other solution to store Bitcoins, such as **tamper resistant card** or **online wallet**. In the latter case, we give our keys to a trust entity, which acts like a bank and stores our Bitcoins. Another solution is to store Bitcoins in a **hardware wallet**. It represent a robust approach, but the users face the risk of losing access to their tokens if they misplace or forget their keys. A way to increase the security of the previous approaches is to use a **secret sharing mechanism**. Its main idea is to divide our secret key into some number N of pieces and store them in several places, in such a way that if we are given any number K of those pieces then we will able to reconstruct the original secret, but if we are given fewer than K pieces then we won't be able to learn anything about the original secret. For example, suppose we have $N = 2 = K$. This means that we are generating 2 shares based on the secret, and we need both shares to be able to reconstruct the original secret. Let's call our secret S which is a 128-bit number. We could generate a 128-bit random number R and make the two shares be R and $S \oplus R$, i.e. we have use OTP schema. Next, we store the previous two shares in separate places. We can observe that neither the first nor the second share by itself tells us anything about the original secret S. Instead, if we have both shares we can simply XOR them in order to reconstruct the original secret. An interesting observation is that this trick works as long as N and K are the same, because we would just need to generate $N-1$ different random numbers for the first $N-1$ shares and the final share would be the XOR between the secret and all the others $N-1$ shares. This means that if N is

greater than K, this doesn't work anymore and we need to use the algebra. For example, when $K = 2$ with $K < N$, then we can represent such schema in the 2D plane as a line with equation $y = k_1 x + k_2$, where k_1 and k_2 represents the two shares. This means that if we know just two points of the line we can simply find the two shares, and thus we will be able to reconstruct the original secret. Naturally, this approach can be generalized in order to require $K > 2$ shares in order to be able to reconstruct the original secret.

There are several kinds of entities that trades Bitcoins against flat currency like dollars and euros. In particular, we have :

- **Market users** : they are users that wants to use Bitcoins for privacy reasons. For example, they buy Bitcoins to perform transactions and then sell them.
- **Investors** : they are users or companies that buy BTC to make money, with the hope that their value will increase for a possible future reselling.
- **BTC intermediaries** : they are entities that simply sell and buy BTC, earning money by exchange fees.

The Bitcoin main advantage is **anonymity** but this is also a disadvantage, because allows both benign and malicious users to be anonymous. The price of Bitcoin will be set by supply and demand : the supply of Bitcoins that might potentially be sold and the demand for Bitcoins by people who have dollars/euros. Naturally if the demand decreases then the Bitcoin value will decrease as well, and increases in the other way around. Some economic models showed that the Bitcoin value depends also on the supply of BTC.

5.4 Bitcoin anonymity

For the anonymous word there can be two interpretations : without using our real name, and without using any name at all. Indeed in Bitcoin each user is specified by a public key which we typically call its **pseudo-identity**. The idea is that we do not know who physical person is behind that public key, but we know that over time the same person behind that public key will be the same person. However, anonymity is not sufficient for privacy but it's necessary. The problem is that in practice anonymity is unachievable. For this reason we will define anonymity as a combination of two aspects : pseudonymity and unlinkability. The first one we know what is, it's just the identity of a user represented by a public key. The second one expresses the

fact that different interactions of the same user with the system shouldn't be linkable to each others. In particular, we reach unlinkability if the following properties holds :

- It should be hard to link together different addresses of the same user.
- It should be hard to link together different transactions made by the same user.
- It should be hard to link the sender of a payment to its recipient.

The first property is not satisfied because if, for instance, Alice uses two addresses to pay for an item, then with a high probability, the input transactions to perform this payment belongs to the same user even if they have different public keys. Neither the second property is not satisfied, because all transactions that have the same public key also belongs to the same user nor the third property is satisfied since by the definition of the protocol, if in a payment there is a remainder, then it will be sent back to the sender. Indeed Bitcoin is linkable, because all the interactions of all users are on the public block chain. So every user can check easily all the activities of any user. Using the previous observations combined with some external information, if available, it's possible to deanonymize the Bitcoin network. In particular, there are three ways to deanonymize the Bitcoin users :

- Even though Bitcoin transactions are randomly transmitted over the peer-to-peer network, this system is not perfect. If, for instance, an attacker is able to connect multiple nodes to the Bitcoin network, he may be able to leverage the information of such nodes in order to understand where a transaction has been originated.
- There are Bitcoin services that requires our real identity, so if we perform a transaction through such services it will be very easy for them to match our pseudo-identity with our real identity.
- The block chain contains all the transactions for all Bitcoin users is public, so that any user can trace all the transactions of any other user.

The "taint" of a Bitcoin transaction evaluates the association between an address and earlier transaction addresses. The more "taint" the stronger the link is between the two addresses. The most common example of user deanonymization is the deanonymization of silk road, that was the largest

market for illegal drugs based on a Tor hidden service and Bitcoin as currency. Even if the founder of such website tried to cover his traces, he was arrested two years after the birth of its web site, because police started to trace transactions correspondent to silk road, until they reached the conclusion that he was the owner. Naturally there are good and bad points in anonymity. The good point for sure is that being anonymous some other user can't learn anything about our Bitcoin balance. However, anonymity doesn't avoid criminals to perform money laundering. For this reason, in several countries the usage of Bitcoins is restricted or forbidden. How can we make Bitcoin anonymous ? A possible approach is to use a technique called **mixing**. It allows to mix several services to mix one's fund with other people's money, intending to confuse the trail back to the funds original source. In traditional financial systems, the equivalent would be moving funds through banks in countries with bank secrecy laws (Cayman islands, Bahams, etc). When mixing Bitcoins, we send our money to an anonymous service and it will send us someone else's coins. So, now, whatever those coins were used for may now be traceable back to us. Furthermore, mixing large amounts of money may be illegal. There are some dedicated mixing services which promise not to keep records, don't ask for our identity, swaps an user's coin (address) with other users's coins (addresses) and has a relatively small anonymity set, because only comprise of those users who use the mixing service at that time instant). The problem is that mixing services may themselves operates with anonymity. Thus, if the mixing output fails to be delivered or access to funds is denied there is no recourse. So an user before using such type of services needs to trust them analyzing for example their reputation, which are the cryptographic warranties, etc. In order to increase unlinkability we can use the mix chain, i.e. a series of mixing services to make a longer chain that makes more difficult to track the transactions between two users. However, the main limitation of this approach is that all the mixed transactions must have the same value. An alternative solution is to have a **decentralized mixing** (e.g. Coinjoin), where different peers self-organizes in order to create the transaction. This approach offers several advantages such as absence of bootstrapping problem, theft impossible, possibly better anonymity and good philosophically alignment with Bitcoin. The main limitations of this approach is that the finding peers is not so easy, any contacted peer will learn the input-output mapping of a transaction and possibility to be subject to a DoS attack. We can apply the Strawman solution in order to solve the second limitation. In particular, first of all we need to exchange the inputs, then we disconnect and reconnect over Tor in order to hide our identity and only after that

we exchange the output. For solving the last issue has been proposed the following solutions :

Proof of Work (PoW) In a PoW system the network participants have to solve the so called **cryptographic puzzles** to be allowed to add new blocks to the blockchain. Typically this puzzle-solving process is commonly referred to as mining. Because the input of each puzzle becomes larger over time, the PoW mechanism requires a huge amount of computing resources, which consume a significant amount of electricity. If a network participant solves a cryptographic puzzle, it proves that he has completed the work and is rewarded with a digital form of value. This reward serves as an incentive to uphold the network. Indeed the Bitcoin is based on a PoW consensus mechanism. There are also other cryptocurrencies that uses this approach such as Litecoin, Bitcoin Cash, etc.

Proof of Stake (PoS) In a PoS system a transaction validator (i.e. a network node) must prove ownership of a certain amount of coins in order to participate in the validation of transactions. This act of validating transactions is called **forging** instead of mining. For example, in the case of cryptocurrencies, a transaction validator will have to prove his "stake" of all coins in existence to be allowed to validate a transaction. Depending on how many coins he holds, he will have a higher chance of being the one that validate the next block, because if he has a greater seniority within the network wrt the other nodes and this fact gives him a more trusted position. Next, the transaction validator is paid a transaction fee for his validation services by the transacting parties. There are some cryptocurrencies that uses a PoS consensus mechanism such as Neo and Ada (Cardano).

Permissionless vs Permissioned blockchain On an open **permissionless blockchain** a person can join or leave the network when he want without having to be pre-approved by any central entity. There is no central owner of the network and software and identical copies of the ledger are distributed to all the nodes in the network. It's not a chance that most of the cryptocurrencies currently in circulation are based on this approach (e.g. Bitcoin, Bitcoin cash, Litecoin, etc). Instead on a **permissioned blockchain**, the nodes have to be pre-selected by a network administrator to be able to join the network. This allows to easily verify the identity of the network participants, but at the same time it also requires network participants to trust the central authority (network administrator) to select

reliable network nodes. In turn permissioned blockchains can be divided into two subcategories : **public permissioned blockchains** and **private permissioned blockchain**. The first ones can be accessed and viewed by anyone, but where only authorized network participants can generate transactions and/or update the state of the ledger (e.g. used in Ripple, Neo). In the latter ones the access is restricted and only the network administrator can generate and update the state of the ledger.

5.4.1 Zerocoin and Zerocash

Zerocoin and Zerocash cryptocurrencies has been introduced to increase anonymity. Zerocoin is a protocol-level mixing, i.e. the mixing capabilities are embedded into the protocol itself. Unfortunately, it's not currently compatible with Bitcoin. If we hold a Bitcoin, we can mint a Zerocoin and then spend them anonymously. In particular, the underlying crypto relies on zero-knowledge proofs and commitment schemes. In order to spend a Zerocoin S, we need to prove in zero-knowledge that we own that coin and it hasn't been already spent. This proof is efficiently verifiable, even if it doesn't leak any other information. Zerocash is a much more efficient system (uses SNARKs) for anonymous payments from Bitcoin. The ledger stores the existence of transactions and the e-cash is untraceable. The only problem is that this requires a setup of public parameters, starting from random secret inputs : if you own such secrets, you break the system.

5.5 Forks

A fork is related to the fact that different parties need to use common rules to maintain the history of the blockchain. When parties are not in agreement, alternative chains may emerge. In general, we can distinguish between two types of fork :

- **Hard fork** : it's a rule change such that the software validating according to the old rules will see the blocks produced according to the new rules as invalid. In this case, all nodes meant to work in accordance with the new rules need to upgrade their software. If one group of nodes continues to use the old software while the other nodes use the new software, a permanent split can occur (we will have a new coin called **split coin**). We can think about a hard fork as an expansion of the rules. Nodes that continue running the old version of the software will see the new transactions as invalid. So, to switch over to the new chain and to continue to mine valid block, all of the nodes in

the network need to upgrade to the new rules. The problems comes when there are some sort of political impasse that push a portion of the community to stick by the old rules, and the data and ruleset are still perceived to have value, meaning that miners still want to mine a chain and developers still want to support it. When the hard fork happened each user of Bitcoin following the new rules got 1 unit of the split coin.

- **Soft fork** : it's described as a fork in the blockchain which can occur when old network nodes don't follow a rule followed by the newly upgraded nodes. This is why soft forks need a majority of hash power in the network. This could cause old nodes to accept data that appear invalid to the new nodes, or become out of sync without the user noticing. We can see a soft fork as any change that is backward compatible (hard fork opposite). When a soft fork is supported by only a minority of hash power in the network, it could become the shortest chain and get orphaned by the network, or it can act like a hard fork.

Now, we want to discuss the so called **transaction throughput problem**. We know that the number of the transactions on the Bitcoin blockchain has increased up to $400k$ transactions per day. Every time an user sends a Bitcoin transaction, such transaction is added into the **memory pool**, which is essentially a pool of all unconfirmed transactions in the Bitcoin network. From the memory pool, miners select transactions that they want to verify. Once miners validate a transaction, they add it to a new block, which is eventually published to the blockchain. Other nodes then iterate through this newly published block's transactions to ensure the block is valid, before accepting the block as a part of its ledger. A simple transaction has size approximatively equal to 226 bytes. A block's size is limited to 1 MB and it's published in the blockchain once every 10 minutes on average. Hence the number of transactions every 10 minutes is at most $\frac{1000000}{600 \cdot 226} = 7.37$ per second. If the memory pool contains several unconfirmed transactions, how does a miner select which transactions to verify ? The sender of a transaction has the option of adding a custom transaction fee for the miner, incentivizing a miner to select the transaction and have it verified faster. Miners will select the transactions that have the highest fee attached to them to maximize profits. The problem is that if Bitcoin does change transaction fee will increase and people will leave Bitcoin. A possible solution is to increase the block size from 1 MB, in order to allow more transactions per block. This can be done if all miners agree therefore requiring consensus from the Bitcoin community. However, there are million of people that

uses Bitcoin, so reaching consensus is a very hard task. Since the Bitcoin user base continues to grow, there will be eventually another backlog of unconfirmed transactions, so another block size increase will be needed, and subsequently another hard fork. Another possible approach is to allow blocks to have any size. However, this introduces several issues because handling a big block is impractical and will restrict mining even more introducing an element of centralization. Furthermore, once a block is mined, all the other nodes must validate the block before accepting it. If the block size is incredibly large and somebody were to publish an invalid block, nodes would waste a large amount of time attempting to validate the block before discarding it as invalid and moving onto the next block. This allows a DoS attack by repeatedly publishing insanely large invalid blocks to the network, stopping valid blocks from being processed for a long time period. Satoshi was able to make numerous changes to the Bitcoin network at the beginning, but now make changes is way more difficult.

5.6 Other digital coins

Due to the scalability issue of the Bitcoin network, new digital currencies started to arise. Some of the most commonly used are reported below.

Ethereum Ethereum is a digital currency born in 2015 and its main characteristic is the ability to run "smart contracts". Smart contracts are "self-executing" contracts or applications that run exactly as programmed without any possibility of downtime (i.e. the blockchain is always running), censorship, fraud or third-party interference. Ethereum has a capability that goes far beyond that of a pure peer-to-peer digital cash equivalent like Bitcoin. In simple terms, it's much like a smartphone operating system on top of which software applications can be built. Technically speaking, the Ethereum platform itself is not a cryptocurrency and is a permissionless blockchain. However, it requires a form of on-chain value to incentive transaction validation within the network, i.e. a form of payment for the network nodes that execute the operations. This is where the Ethereum's native cryptocurrency **ether (ETH)** comes into play. Ether doesn't only allow smart contracts to be built on the Ethereum platform, but it also functions as a medium of exchange.

Ripple Ripple is an open-source P2P decentralized digital payment platform that allows for fast transfer of currency, based on a public permissioned blockchain. First of all a company needs to receive a "BitLicense" for an

institutional use case of digital assets from the New York's Department of Financial Services. It's also getting support from a number of big players in the financial services industry such as Santander, Bank of America Merill Lynch, etc. Ripple uses the cryptocurrency **XRP**, which was built to allow financial institutions to settle cross-border payments a lot faster and cheaper than they can using the global payment networks that are in place today, which can be slow and involve multiple middlemen (i.e. banks). According to Ripple, XRP can handle more than 1500 transactions per second. Ripple is not based on a PoW or PoS mechanism to validate transactions, but it makes use of its own specific consensus protocol. The total supply of XRP has been created upon the coin's inception by its inventors.

Stellar

Bitcoin Cash

Litecoin

IOTA

5.7 Algorand

The are two different needs in a blockchain :

1. The first is making the blockchain tamper-proof. This has been solved via the usage of one-way hashing function : the hash of the latest block is included as part of the next block. All blockchains shares this approach, so they are all equal under this aspect.

2. The second need is generating new blocks : how can we select a new block to be appended to the block chain ? This is the real challenge, and the blockchains mainly differs in how they accomplish this task. A new block should contain a set of valid transactions that do not appear in the blockchain so far. The problem is that at any point in time two users may have seen the same sequence of blocks but may see different new transactions. This is the case because, in a distributed ledger, each transaction is not instantaneously propagated through the network. Bitcoin solves this problem using the PoW mechanism.

Blockchain Trilemma An existing blockchain can offer at most two of the following the properties :

- **Security**
- **Scalability**
- **Decentralization**.

In fact, Bitcoin has two problems : PoW doesn't scale and it's very slow; in fact the crypto puzzles are so hard in order to guarantee that one solution is found only every 10 minutes, no matter on how many miners try to solve the crypto puzzle. This time is not compatible with an increased us of BTC. The other problem is that PoW results in De-Facto centralization. Indeed, PoW has caused a huge concentration of power. This centralization is a consequence of the fact that PoW is both expensive and wasteful. Mining today utilizes racks and racks of specialized hardware and consumes an enormous amount of electricity.

Now, lets talk about the Algorand protocol which has been designed by Silvio Micali. The key ideas of Algorand is that the users are weighted by the money in their account and instead of having all the nodes run a Byzantine Consensus algorithm (BA), let a subset of nodes represent the whole group and run BA. In particular, the idea is that when a node propose a new block a committee is chosen at random, which runs the consensus algorithm. In other words, at random is chosen an user to propose a new block and there is a committee that validates it. Suppose that a malicious user has been chosen. So he is able to propose a new block that contains, for instance, double spending transactions. However, the block is accepted if and only if the committee decides to accept the block, and thus the attacker can be caught assuming that the majority of the people in the committee are honest. In Algorand a new block is constructed in two phases :

1. In the first phase a single token (token = one unit of digital currency) is randomly selected, and its owner is the user who proposes the next block.
2. In the second phase 1000 (not a fixed constant) tokens are selected among all tokens currently in the system. The owners of these 1000 tokens are selected to be part of a phase-2 committee, which approves the block proposed by the first user if valid.

In the second phase if the majority of users are honest then the second phase will guarantee an agreement among honest users. Now the questions are the following : How can we select at random the committee members and ensure that an adversary cannot fake being a committee member ? This is done by running a "cryptographic lottery" on each user device. When an user wins the lottery, distributes its winning ticket in the network such that everybody can verify that such user is a winner. If there aren't any problems, such user will be part of the committee. Obviously, any user cannot cheat or modify the lottery in such a way to increase the chance to win beyond the number of tokens that the user have. Since the number of users that forms the committee is typically small, the committee members communicate via gossip. In particular, each user collects a block of transactions he hear about. Algorand will initiate a round starting with a block proposal, in which each committee member will propose its block. Users will wait for a time period to receive the blocks and keep only the one with the highest priority. At the end, all users who received some block will start the consensus algorithm to reach the majority of votes and commit a block. This approach is scalable since, once an user of the committee has been selected, each member propagates to the network a single message. So no matter how many users are in the system, the maximum number of messages that needs to be propagated is small. Furthermore, this approach is also secure, because if an attacker wants to corrupt the committee members, he can't since he doesn't know who they are. So corruption is not possible, since whatever the committee members had to say, they have already said propagating their winning tickets and opinions about the block. In the next round, different committee members will be chosen. The potential weaknesses of this protocol are the gossip algorithm used by committee members to talk between each other which became very expensive in presence of a huge number of tokens, it's limited to cryptocurrency and for sure we need to take into account the tradeoffs between public and private blockchain.

6 k-Anonymity and other cluster-based methods

We know that a large amount of person-specific data has been collected in recent years from both governments and private entities. The information extracted from such data are used for analyze trends and patterns or to formulate public policies. There are laws and regulations that requires that some collected data must be made public (e.g Census data). The problem is that we want to publish these data because they might have an impact, for example Health-care datasets, but at the same time we want to preserve anonymity, i.e. such data must be used only for research purposes but they should not be used for other purposes. This aspect goes under the name of **inference control**. The idea is the following : we have individuals that submit data, these data are collected and we apply to them a masking process in order to remove the possibility of identify the people, and finally we publish the modified data so that researchers can use them. In particular in the masking process we have two conflicting goals : on one side we want to protect the confidentiality of individuals, on the other side we would like to preserve the utility of these data reducing as much as possible the information loss. First of all we would like to eliminate the person identifying information such as name, phone number, email, etc. However, this is not sufficient because we might release information using some additional information. The data we are talking about are also called **microdata**, and the resulting masked data are called **masked microdata**. Instead, all the computed data, for instance for statistics purposes, are called **macrodata**. The problem is that even if we do not publish the individual data, there may be some fields that may uniquely identify some individual, which an attacker can leverage them to identify to detect a specific person. The same reasoning is applied for macrodata. This process is also called **re-identification by linking**. For this reason, in order to protect macrodata, we might suppress some cells by removing their value. These are called **primary suppression**. However, this doesn't solve the problem because an attacker could guess with a certain probability the suppressed cell computing an interval that contains the target cell leveraging the marginal totals of the data. To block such inferences, additional cells may need to be suppressed (**secondary suppression**) to guarantee that the interval are sufficiently large, and thus to reduce the probability of the attacker to make the right guess. A **quasi-identifier** is an identifier that allows to uniquely identify a certain amount of the population. For example, the zip code, birth date and gender uniquely identify the 87% of the population in the U.S. To solve this problem we need to introduce the **K-Anonymity** con-

cept. Its main idea is that when we publish some database, the information for each people contained in the released table cannot be distinguished from at least $k-1$ individuals whose information also appears in the released table. In other words, any quasi-identifier present in the released table must appear in at least k records. Naturally these k record form an equivalence class. We can achieve K-Anonymity using two techniques : generalization in order to replace quasi-identifiers with less specific but still semantically consistent values, and suppression to not release a value at all. For example, to apply generalization we might suppress the last 2 digits of the zip code, or the last digit of the age, etc. When we apply the generalization technique, we replace specific quasi-identifiers with less specific values until we get k identical values. While we apply the suppression technique, when the generalization approach causes too much information loss. A possible problem is called **curse of dimensionality** in the sense that generalization that is used fundamentally relies on spatial locality, but real-world datasets has a huge number of attributes. The problem is that we are classified using too many dimensions that is almost impossible to escape out. However a problem that we could have is that some fields could allows the attacker to exclude a priori some of the k-anonymous records. Another problem arose when the record appear in the same order in the released table ans in the original table, allowing an attacker to perform an **unsorted matching attack**. A possible solution is simply to randomize the order before releasing. Another problem comes from the fact that different releases of the same private table can be linked together to compromise K-Anonymity allowing an attacker to perform a **complementary release attack**. An attacker could perform a **homogeneity attack** when a 3-anonymous table contains a field that has the same value for some group of people. We understand that the simple definition of K-Anonymity is not sufficient. A possible solution is to consider the **l-diversity** approach, in which sensitive attributes must be "diverse" within each quasi-identifier equivalence class. However this might not be sufficient, because it doesn't avoid an attacker to perform a probabilistic inference attack. Suppose that only 1% are affected by a specific disease and we have 10000 records. In this case there can be at most 100 people that are affected by such disease. Now, if we are able to make random queries to the database, we might expect that only 1% of the people in the query response will be affected by such disease. Clearly, we have other parameters that should count, and so we might found among those parameters some queries that might allows us to detect a specific person. This kind of attack goes under the name of **skewness attack**. This means that distinct l-diversity might be unachievable in practice, because making more

queries there would be always some category in which there could only few people. We might generalize so much to eliminate all the useful information, but we want to publish useful information that increase our knowledge. The concept of **T-closeness** measure the semantic distance between concepts, requiring the generalization of sensitive information. However this doesn't prevent an attacker that knows some additional information that is not quasi-identifier to identify. K-Anonymity and the other approaches we discussed before are not useful at all, because they focuses on data transformation, but not on what can be learned from the anonymized dataset. Furthermore, it's subject to the curse of dimensionality problem and several attacks discussed before. The K-Anonymity good points are that it's very simple to implement and there are a wide variety of algorithms for implementing it. The major advantages of l-diversity are that it provides a greater distribution of sensitive attributes with the group increasing the data protection, and provides protection against attribute disclosure (it's an enhancement of K-Anonymity technique). However, we already said that it can be unachievable in practice and it's subject to several attacks such as swekness and similarity attacks. Finally, the T-closeness major benefits are that it interrupts attribute disclosure to protect data privacy, protects against homogeneity and background knowledge attacks and identifies the semantic closeness of attributes. However, it also has some disadvantages such as necessitates that sensitive attributes are spread in the equivalence class to be close to that in the overall table and using EMD measure it's very hard to identify the closeness between t-value and the knowledge gained.

6.1 HIPAA privacy rules

These rules have been proposed in the US. The idea is that to publish statistical health data only for legal and economic issues. For example smoking and probability of cancer, etc. In particular, these rules says that the covered entities must remove all of a list of 18 enumerated identifiers and have no actual knowledge that the information remaining could be used, alone or in combination, to identify a subject of the information. The identifier that must be removed include direct identifiers such as name, street address, SSN, birth date, zip code, etc. They allows the initial three digits of a zip code if the geographic unit formed by combining all zip codes with the same initial three digits contains more than 20000 people. In addition, age (if less than 90), gender, ethnicity and other demographic information may remain in the information. They are intended to provide to covered entities with a simple, definitive method that doesn't require much judgment by the covered entity

to determine if the information are correctly de-identified. However, their main disadvantage is that do not guarantee us to reach K-anonymity.

6.2 UK confidentiality guidance

UK in order to ensure the privacy of sensitive data started to use a different approach wrt to HIPAA privacy rules. First of all we try to understand the users requirements for the publication of data. Then we try to understand the key characteristic of these data and if there would be the possibility of disclosure of such data. We need to take into account which are the dangers that can be done by attacker leveraging the disclosure of such data. If required, we select an appropriate disclosure control methods to manages this risk. The idea is that this process is not purely syntactic because we need to do some reasoning on the data. For instance, suppose there is an attacker with a special interest in statistics discover from a table that only a small number of very young women live in a particular area. The small number in the cell doesn't tell the intruder who the women are, but it may leverages other sources of information to locate the individuals and disclose more details. This situation may occur when small values are reported for particular cells. In a large population the effort and expertise required in order to discover more details about the statistical unit may be very high. Naturally, as the population moves to smaller geographical area it becomes easier to find units and discover information. In particular, the risk categories are defined in terms of the likelihood of an attempt to identify individuals, and the impact of any identification. We can distinguish the following categories :

- **Medium risk** : it will be sufficient to consider all cells of size 1 or 2 unsafe. Care should be also taken where a row or column is dominated by zeros.
- **High risk** : the likelihood of an identification attempt will be higher and the impact of any successful identification would be great. All the cells of size 1 to 4 are considered unsafe and care should be taken where a row or column is dominated by zeros. Higher levels of protection may be required for small geographical levels or for particular variables with an extremely high level of interest and impact.

A possible approach to minimize the number of unsafe cells and preserve the original counts is called **table redesign**. Its main idea is to group the categories within a table aggregating to a higher geographical level or

for a larger population subgroup and to aggregates tables across higher geographic level (e.g. zip code) or a number of years/months (e.g. age). Another approach is the **cell values modification** schema, which provides the following features :

- **Cell suppression** : in this way unsafe cells are not published.
- **Rounding** : it adjust the values in all cells in a table to a specified base. This creates uncertainty about the real value for any cell.
- **Barnardisation** : it adjust the cells of every table by $+1, 0, -1$ according to the probabilities.

Another approach is to adjust the data by swapping pairs of record within a micro-dataset that are partially matched to alter the geographical locations attached to the record, but at the same time leave all the other aspects unchanged.

7 Privacy-preserving data mining

An user makes query to a database. He is allowed to make statistical queries and is smart so that is able to infer information he should not know by repeatedly sending queries to the database. Our goal is to allow him to query the database but we want also to preserve the privacy of the database itself. To do that we will preprocess the database records using the **data sanitization** approach. The idea is to overwrite the information in the database with realist looking but false data, generated in a random way, of similar type. Some example of sanitization methods are to simply add random noise to the information retrieved from the database, reveal only summary statistics, output perturbation as in the previous case. We say that this process of sanitization is good if the information that we are asking are drawn by an unknown distribution, because in this way sanitization reveals only the distribution. Suppose that we use a sanitization process that adds a random noise to the data retrieved from the database. The problem is that an attacker might collect several responses asking always the same query to the database in order to leverage them to infer the real answer. So, it's not so easy to define the concept of privacy. Let's see some definitions :

- **Weak** : it means that no single database entry has been revealed.
- **Stronger** : it means that no single piece of information is revealed.
- **Strongest** : the adversary beliefs that the data have not changed.

For the latter case, there is a way to formalize the difference between two probability distributions which goes under the name of **Kullback-Leibler distance**. It's defined in the following way

$$D_{KL}(P \mid\mid Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}.$$

If the distance is small then the two distributions are similar. It measures mutual information between the original and randomized databases. In particular, we will have a privacy breach if the database revealing some answer to a query significantly changes the attacker's probability that $Q(x_i)$ is true, where x_i denotes a record in the database and $Q(.)$ represent some property. In general, let $Q(x)$ be some property and ρ_1, ρ_2 be two probabilities such that the first is negligible and the second is non negligible, we can define two types of privacy breach :

- **Straight privacy breach** : $P(Q(x)) \leq \rho_1$, but $P(Q(x) \mid R(x) = y) \geq \rho_2$, i.e. $Q(x)$ is unlikely a priori, but likely after seeing the randomized value of x.

- **Inverse privacy breach** : $P(Q(x)) \geq \rho_2$, but $P(Q(x) \mid R(x) = y) \leq \rho_1$, i.e. $Q(x)$ is likely a priori, but unlikely after seeing the randomized value of x.

We can formalize our problem by defining for each entry of the database a function such that each entry has an a priori belief and a posteriori belief after seeing a certain numbers of responses provided by the database. If the confidence on the responses doesn't changes at all, why do we make queries ? Instead if the confidence on the responses changes then the attacker is able to extract useful information from the database. In particular, there is an important theorem defined by Dinur and Nissim that says : let n be the size of the database. If $O(n^{\frac{1}{2}})$ perturbation are applied, an adversary can extract the entire database after $poly(n)$ queries, but even with $O(n^{\frac{1}{2}} \log n)$ perturbation, it's unlikely that user can learn anything useful from the perturbated answer (too much noise). The problem is that to implement the strongest notion of privacy is too complex in practice. For this reason we need to work with a weaker privacy definition also called **differential privacy** that shows that it's possible to get positive results.

7.1 Differential privacy

Differential privacy states that whatever is learned about one respondent A would be learned regardless of whether or not A participates to the database. The idea is that the statistical outcome is indistinguishable regardless whether a particular user is included in the database. Formally, for every pair of databases D_1 and D_2 that differs in one row, for every output O, if an algorithm A satisfies differential privacy then $\frac{Pr[A(D_1)=O]}{Pr[A(D_2)=O]} < \exp^{\epsilon}$, with $\epsilon > 0$. In other words, an adversary should not be able to use output O to distinguish between any D_1 and D_2. Naturally we would like that such condition holds for all outputs, because we allow the user to make several queries to the database, and to be sure that he doesn't not learn anything. Suppose that our database has n records. We can define $n+1$ games where game 0 represent the fact that the adversary interacts with the sanitized database when all the data are present, and game i the fact that the adversary interacts with the sanitized database when all the data are present except for the record i. Next, the adversary makes a lot of queries and obtains some transcript S. At this point suppose that we modify the database changing just one record. The adversary will continue to send queries to this modified database and will get a transcript $S^{'}$. The two set of answers S and $S^{'}$ doesn't allow the adversary to distinguish any significant information

about the presence or absence of the record previously changed. In formal way, we will say that the sanitization process San is ϵ-indistinguishable if $\forall A, \forall DB, DB^{'}$ which differ in one row, $\forall S$ we have

$$P(San(DB) \in S) \in (1 \pm \epsilon)P(San(DB^{'}) \in S).$$

This means that indistinguishability implies differential privacy. Indeed, we can say that San is safe if for all prior distributions on DB and for all transcripts S the statistical distance between any pair of a posterior probability given S is less or equal than ϵ, i.e. their distributions are so close that we are not able to distinguish between the two. To implement this approach in practice we simply adds a noise to the sanitized answers such that each answer doesn't leak too much information about the database, but the noisy answers are close to the original answers so that they are useful to the user. Naturally, privacy depends on the noise that we decided to use. The larger the noise is the higher is the privacy and less useful an answer will be. This means that the noise value strictly depends on the records in the database. In particular, we will define the **global sensitivity** measure, which simply measure the differences in the data. There is a well known theorem claiming that if we adds to the real answer a noise generated from a Laplace distribution that is strictly depends from the global sensitivity measure and inversely proportional to ϵ that the resulting answer is ϵ-indistinguishable. Now the question is how much noise do we need for privacy ? We can define it in formal way through the **sensitivity** measure. Consider a query $q : I \rightarrow R$ and be $S(q)$ the smallest number s. t. for any neighboring tables $D, D^{'}$ we have $\mid q(D) - q(D^{'}) \mid \leq S(q)$, If the sensitivity of the query is S, then the following guarantees ϵ-differential privacy $\lambda = \frac{S}{\epsilon}$. The sensitivity metric measures the maximum deviation of a single database entry. However the problem is always the same, the higher the noise is the less interesting the answers will be (e.g. issues for count, sum, max and multiple queries). Another problem is that when we have non-numeric valued query, in which way should we answer ? The intuition is that we need to introduce some randomization in order to return only high quality elements, which are the ones close to the real answer. This approach guarantees privacy, but we know that error in the quality is given by the variance of the dataset. Instead in the Laplace mechanism the variance is much smaller since it doesn't depend on the size of the input. So, this approach may be good for count queries, but in general the Laplace mechanism has a lower variance and thus provides us better results. Now, lets consider a comparison between one-query against multiple-query. Let $\epsilon_1 = 0.1$ be the privacy parameter. In the one-query scenario, the data analyst perform a single query. This means

that the analyst would not be able to perform a second query over the data without risking a breach of the policy. In the multiple-query scenario, the data analyst performs a single query with ϵ_1 privacy parameter. Then the analyst can also perform a second query say with $\epsilon_2 = 0.2$. So, after the second query the overall privacy loss is $0.1 + 0.2 = 0.3$. The problem is that every time the analyst make a query he can get some information about the data, and making many queries he gets many information. So, this poses the fact that the number of queries that should be done must be bounded. Indeed a vast majority of records in a database of size n can be reconstructed when $n \log n^2$ queries are answered by a statistical database, but even if each answer has been arbitrarily altered to have up to $O(\sqrt{n})$ error (Dinur Nissim result). In general, we can distinguish between two kinds of query composition :

- **Sequential composition** : if $M_1, ..., M_k$ are algorithms that access a private database D such that each M_i satisfies ϵ_i-differential privacy, then running all k algorithms sequentially satisfies ϵ-differential privacy with $\epsilon = \epsilon_1 + ... + \epsilon_k$.
- **Parallel composition** : if $M_1, ..., M_k$ are algorithms that access a private database D such that each M_i satisfies ϵ_i-differential privacy, then running all k algorithms in parallel satisfies ϵ-differential privacy with $\epsilon = \max\{\epsilon_1 + ... + \epsilon_k\}$.

To solve the previous problem we can use the following approaches :

- **Direct** : the output of the algorithm is the list of query answers.
- **Synthetic data** : the algorithm constructs a synthetic dataset $D^{'}$, which can be queries directly by the analyst, even though answers may not be accurate.

7.1.1 Deploying differential privacy for 2020 Census

The laws in the US requires that everything should be counted only once and in the right place. There are quite established rules about which data should be collected and made public (e.g. race/ethnicity, occupancy status and group quarters population). The data that should be collected must preserve individuals, neither the secretary nor any other officer may get information about such data. The data are collected only for statistical purposes, kept confidential and in any way they won't identifies individuals or establishments. Our goal is to avoid improper disclosure of the data. For

instance the 2010 Census collected data on 308 million people, and the process for preserving privacy is the following : the raw data are collected from respondents, then the unduplicated data are selected and there is a process that implements some operations on the data in order to alter them and guarantee their confidentiality. Next the statistical data are computed and organized in tables. The problem is that publishing too many statistics result in the compromise of the entire database of confidential data, because we can setup a system of equations to reconstruct a significant portion (about 38%) of the original data. However, this kind of attack is not perfect because the attacker would not know which re-identifications are correct. To solve this problem statistical agencies have just two choices : publish fewer statistics or publish statistics with less accuracy. The first option is not allowed because if we publish fewer statistics then we don't know when we have reached a threshold of having published too many statistics. This is because the information that is published can be combined with external information. We can just use the second option which leverages the differential privacy, to get a tradeoff between data accuracy and privacy loss. An important point is that if we go through the "noise barrier" several times for the same data, probably we will get different numbers. Another important aspect is that the noise has also an impact on the number of people that are on the statistics. For example, if there are more people then we can put a small error to avoid re-identification attacks. Whereas if we have just few people then the error should be high. The advantages of such approach are the following : the details can be explained to the public, we can tune the error parameters in order to reach different privacy levels, the privacy guarantees don't depends on external data, we are protected against re-identification attacks and protects every member of the population. However, we have also some challenges to face such as entire country must be processed at once for reaching the best accuracy, and every use of confidential data must be taken into account the privacy-loss budget. So we assume that every record in the population may be modified, but there is a global privacy budget that should be kept constant. This means that records in the tabulation data have no exact counterpart in the confidential data. The entire census is treated as a set of queries on histograms. For each geographic level (e.g state, country, etc.) confidential data is used to build the corresponding public histograms. Next, these histograms are post-processed to guarantee their consistency. The main issue in the US Census is the size of the blocks, because we know that for blocks with a small size we can build a system of equations to obtain information about people. A possible approach is to independently protect each block, with which the measure queries and the

reconstruction is done on each block. The main advantages of this approach are the following : it simple and easy to parallelize, privacy cost doesn't depends on the number of blocks and releasing differential privacy for one block has the same cost as releasing it for all blocks. However, we will have significant error at higher level and the variance of each geographical unit is proportional to the number of blocks it contains. For this reason, another possibility is to use the top-down mechanism, which first generates national histogram without geographic identifiers and then allocates counts in histogram to each geography "top down". This approach is easy to parallelize, each geographical unit can have its own strategy selection, we have a parallel composition at each geo-level and we have also reduced the variance for many aggregate regions. To summarize what we have said so far we can say that :

- The fundamental law of information reconstruction tells use that if we publish too many statistics derived from a confidential data source with a high degree of accuracy, then after a finite number of queries we will completely expose the confidential data (Dinur and Nissim result). All the statistical disclosure limitation techniques tries to protect privacy by limiting the quantity of data released or by reducing the accuracy of the data itself.

- When implementing differential privacy, the privacy-loss budget makes data accuracy and privacy competing uses of a finite resource (bits in the underlying data). It's impossible to protect privacy while also releasing highly accurate data to support every use case, and vice versa. The statistics for large populations can be protected with a negligible amount of noise, whereas to protect the statistics for small populations is required a significant amount of noise.

- How we implement any disclosure avoidance strategy will impact the accuracy and usability of the resulting data. Indeed, the design of the system can often have a greater impact on the accuracy of the resulting data than the selection of the privacy-loss budget. With differential privacy, the amount of noise we must inject into the data strictly depend on the sensitivity of the calculation we are performing, because such sensitivity depends on the impact that the presence or absence of any individual could have on the resulting calculation. For this reason, some statistics typically require less noise than others.

- We need to take into account that to guarantee differential privacy different tabulations of the same characteristic may not be internally

consistent, because typically differential privacy inject noise from a symmetric distribution including fractional and negative values. However, converting such noisy values into non negative integers to guarantee consistency introduces more error into the data that is strictly necessary to protect privacy.

7.2 Netflix prize dataset

Netflix has started as a company that was online movie rental service. Indeed Netflix instead of renting movies start to deliver them on the Internet. In October 2006, it released real movie ratings of $500k$ subscribers because they have a good recommendation system but they would like to improve it. So, they make a prize publishing data about 10% of all Netflix users, clearly removing names, perturbing rating information and taking the average user rated over 200 movies. The goal was to predict how a user will rate a movie beating the Netflix's algorithm and gets a 1 million prize. Analyzing the data, we see that the most popular movies were rated by almost half of the users, while the least popular just by few users. However, two researchers was able to deanonymize the released database. The idea is of fixing some target record r in the original dataset, and the goal is to learn as much about r as possible by collecting information from the released database. However, the problem is that the background knowledge (IMDB dataset) is nosy, only a small portions of the samples has been releases, there may be false matches and the records itself may be perturbed. Anyway, since ratings about least popular movies are very personalized (it's not so common that two users give the same rating for a not popular movie), the researchers found that, with this cross references on movie ratings and date of ratings, some users turned out to be members of both IMDB and Netflix and private information was obtained with a very low probability of error. Indeed, in average just 2 ratings on not famous movies are sufficient to reduce the number of candidates to 8, and only 4 ratings are sufficient to uniquely identify the user in the database published by Netflix. The fundamental point is that we needs always to take into account the prior knowledge that an attacker may have to understand if our data really guarantees privacy or not.

7.3 Social network deanonymization

Given a network, we will say that such network is partially or totally deanonymized when the attacker, starting from a re-identification of a fraction of the

nodes from an auxiliary network that partially overlaps with the original network, gains knowledge about the edges of the graph that represent the network (1-1 mapping between two graphs). For instance, we can consider the partial deanonymization of Twitter graph using the Flickr graph. In particular both companies shared mandatory username, optional name and location. Researchers were able to reach roughly 30.8% of the mappings correctly re-identified, 57% were not identified and 12.2% were identified incorrectly. Instead, if an active adversary is a member of a social network could deanonymize it in the following way : first of all he creates fake nodes if needed, and adds edges in the social network graph with features that easily help him to recognize himself in the anonymized version of the graph. Typically, the attacker have access to different network, whose membership partially overlaps the network targeted by the attacker. If it's not the case, the information that the attacker needs might be extracted from the original network with the automatic crawling approach using some malicious third-party application. In positive case, the deanonymization process is performed via a re-identification algorithm, which is constituted by two phases :

1. **Seed identification** : the algorithm identifies a small number of seed nodes in both target network and auxiliary network, mapping them with each other. In order to find this mapping, we assume a clique structure in the auxiliary network and search the same clique on the target network, using common neighbors and the nodes degree.

2. **Propagation** : in this phase the attacker builds a deterministic 1-1 mapping between the nodes of the target and auxiliary network. Each iteration starts with the accumulated mapping, and then an unmapped node u in the target network is picked and a score for each unmapped node v in the auxiliary network is computed. We want to underline that the algorithm finds new mappings using the topological structure of the network and the feedback from previous mappings.

7.4 Privacy concerns in social networks

We know that social networks may appears useful because allows us to connect with our friends, post photos and videos, etc. However, people posting information by themselves on the social networks increase the probability to be subject to malwares, stalking, etc. leading to several privacy issues. All of this information about an user could be useful when combined with

social engineering techniques to start an identity theft attack or to compromise sensitive information. Typically adversaries targets social network users to obtain information to be used in phishing and other types of attacks. Often the users are not aware of the amount of information that is shared in social networks. Another problem is that the facility with which a person can create a fake account on the social networks and to possibly lures other users in doing something wrong in order to compromise their privacy. For example Facebook in January 2012 did an experiment over $600k$ users randomly selected, to determine the effect of the emotional alteration through the interactions of those users with Facebook itself. This is a clear example in which the information of each user to conduct the experiment is seen as a breach of privacy without the user disclosure. Another problem is that users accept people as "friends" that doesn't know at all just to enlarge the "friends group", but at the same time allows such people to access their personal information and pictures. We understand that the social network privacy doesn't exist within the realm of privacy settings, but the privacy control is within the hands of the users. Another interesting fact is that 69% of Facebook users believe that the information about other Facebook users reveal may create privacy risks for those users, but the majority of users think that the information about themselves don't represent a privacy risk at all.

8 Legal issues and GDPR